Facultad de Informática
Universidad Complutense de Madrid

Proyecto de Sistemas Informáticos 2005/2006

EMULADOR DEL MOTOROLA 68000

Autores:
David Bordas González
Raúl Alonso Sáez
Rui Miguel Alonso Da Cruz

Director de proyecto:
Daniel Mozos Muñoz

Los abajo firmantes:

David Bordas González, Rui Miguel Alonso Da Cruz y Raúl Alonso Sáez, autorizan a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales, y, mencionando expresamente a sus autores, tanto la presente memoria, como el código, la documentación, y/o el prototipo desarrollado.

David Bordas González Rui Miguel Alonso Da Cruz Raúl Alonso Sáez

Madrid, a 7 de Julio de 2006.

Indice

1. Introducción. Justificación.

El de LEC es uno de los laboratorios que tradicionalmente más quebraderos de cabeza han venido dando a los alumnos, y, realmente una gran parte de la dificultad de la asignatura radica en el hecho de que para probar las prácticas realizadas en casa es preciso acudir explícitamente al laboratorio a depurarlas sobre el hardware específico.

La masificación en la asignatura respecto al número de equipos disponibles, bastante limitado, hace que incluso tomar sitio en horas libres fuera casi imposible.

Es por ello que se nos ocurrió la idea de desarrollar un software que emulara el comportamiento del motorola 68.000 tanto con los periféricos disponibles en el laboratorio como con el software propio asociado al procesador.

Con este emulador, ya no sólo los alumnos no tendrán la necesidad de acudir a los laboratorios a probar y depurar sus programas, sino que tampoco debería ser necesaria una ampliación del número de equipos, por otra parte costosos y difíciles de adquirir. El sistema simulará tanto el programa monitor como el 68000 y los periféricos, partiendo de ficheros de código objeto s-record previamente generados mediante un compilador de motorola 68000 real.

Además, el hecho de haber sido desarrollada esta herramienta en Java le dota de la potencia de un lenguaje portable. Incluso podría ser utlizada desde el campus virtual por los alumnos que cursaran la asignatura.

Es por ello que creemos que nuestro proyecto, aparte de ser una idea práctica y útil puede ser una herramienta bastante pedagógica y de gran uso docente en la carrera.

En los sucesivos apartados de esta memoria iremos explicando desde una introducción previa al 68000, Programa Monitor e Interfaz de E/S asi como al puesto de trabajo, cómo a través de una organización del modelo en paquetes lo más fiel posible a la realidad hemos sido capaces de emular el correcto comportamiento de las prácticas vistas en el laboratorio**.**

The main aim of this proyect is to provide the LEC students of a Motorola 68000 software emulator so that the necessity of taking presence into the laboratories which the hardware is available was unecessary.

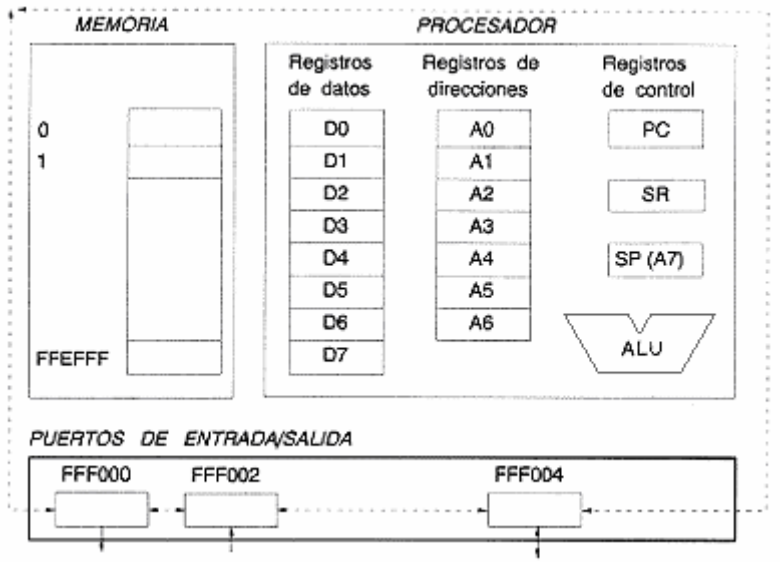
With this tool we emulate not only the 68000 and his software, but all the devices asociated, starting from the s-record code object previously generated by a real 68000 compilator.

2. Motorola 68000

2.1 68000

Esquemático:

El Motorota 68K está dividido en varias partes claramente distinguidas: procesador, memoria y puertos de entrada-salida.



1) Procesador:

- **Di**. Contiene 8 registros de datos accesibles/modificables con las precisiones Byte, Word y Long Word. La simulación de estos registros es total, su implementacion es completamente transparente y contiene exactamente los mismos 0's y 1's que contendrian los registros reales ejecutando los mismos programas.

- **Ai**. También están simulados de forma semejante los registros de direcciones, con la diferencia de que estos registros no son direccionables por Bytes y que el registro A7 está reservado como puntero de pila SP.

- **PC**. El contador de programa es modificable y accesible mediante la consola volcando siempre todo el contenido, ya que solo es seleccionable con precisión Long. Está implementado como un vector de bits al igual que los anteriores registros.

- **SR**. El registro de estado está simulado completamente aunque a lo largo de la aplicación sólo se acceden y modifican los bits referentes a las condiciones *carry*, *zero*, *overflow* y *negative*.

- **ALU**. La unidad Aritmetico-lógica está preparada para recibir uno o dos operandos en forma de vectores de bits y una operación. Como salida devolverá el resultado y generará la actualización del vector de condiciones.

2) Memoria:

La memoria ha sido implementada en lugar de con vectores binarios con cadenas de dígitos hexadecimales, con el objeto de hacer más fácil el acceso, tratamiento y mostrado por pantalla de los contenidos.

Es direccionable por Bytes y cumple todas las restricciones en cuanto accesos y modificaciones se refiere que vienen determinados en el manual oficial del Motorola 68K.

3) Puertos de Entrada/Salida

Los puertos de entrada-salida están destinados a comunicar el Motorola con el mundo exterior. En nuestro caso se encargará de comunicar con la placa de adquisición de datos (la MVME) de la que recibirá las entradas y por la que mostrará las salidas.

Este aspecto se explicará con mucho más detalle en el siguiente apartado.

Obs: Para más detalle ver la implementación Java referente a los registros, la ALU, la memoria y las puertos de Entrada/Salida en el apartado 3.

Funcionamiento:

El comportamiento del 68000 simulado es semejante al del aparato real. Todos los componentes contienen exactamente los mismos bits y el secuenciamiento de las ejecuciones es similar. Se lee el contenido de la memoria apuntado por el contador de programa, se desensambla la dirección para saber que instrucción contiene, se ejecuta la instrucción y se realizan los cambios que provoque dicha instrucción en memoria, registros y vector de condiciones.

2.2 Programa Monitor

El programa MVME101bug 3.n está alojado físicamente en las direcciones F00000-F07FFF de memoria y está provisto de una serie de comandos de entrada/salida ejecutables desde un terminal cuyo comportamiento hemos tenido que emular.

No hemos implementado la totalidad de los mismos, pero si un amplio subconjunto que comprende los más comunmente empleados, así como ciertas mejoras sobre el terminal respecto al real, que lo hacen más funcional y cómodo, como por ejemplo un recordatorio de últimos comandos ejecutados.

El objeto de este apartado es describir dichos comandos, junto con su sintaxis y su funcionalidad.

.A
Muestra todos los Registros de Direcciones

.A0 - .A7 [<data> / <address>]
Muestra / modifica el registro de dirección (A0 - A7) seleccionado donde:

- <data> es el valor numérico a almacenar en el registro
- <address> es la dirección almacenada en el registro

BR [<address> [; <count>]] [<address> [; [<count>]] ...
Establece un breakpoint en <address> con un contador <count> donde:

- <address> es la dirección de memoria en la que el programa parará
- <count> es el nº de pasadas por el breakpoint hasta que el programa para

Usado sin parámetros muestra por pantalla todos los breakpoints existentes

.D
Muestra todos los Registros de Datos

.D0 - .D7 [<data>]
Muestra / modifica el registro de datos (D0 - D7) seleccionado

- <data> es el valor numérico a almacenar en el registro

GO [<address>]
Inicia la ejecución del programa a partir de la dirección especificada donde:

- <address> es la dirección de comienzo de la ejecución. Si no se especifica, comenzará la ejecución en la dirección que haya en el PC

GT <address>
Inicia la ejecución del programa a partir de la dirección del PC estableciendo un breakpoint en la dirección especificada donde:

- <address> es la dirección del breakpoint permanente

HE <command>
Muestra el contenido de la ayuda para el comando especificado donde:

- <command> es uno de los siguientes comandos:
  - .PC .SR .US .SS .A .D .D0 .D1 .D2 .D3
    .D4 .D5 .D6 .D7 .A0 .A1 .A2 .A3 .A4 .A5
    .A6 .A7 .R0 .R1 .R2 .R3 .R4 .R5 .R6 BD
    BF BH BI BM BO BR NOBR BS BT DC
    DF DU G GD GO GT HE IOP IOT LO1
    M MD MM MS OF PA NOPA PF T TM
    TR TT VE DATA ADDRESS

    cada uno de los cuales muestran la información correspondiente a dicho comando en caso de estar implementado, a excepción de DATA y ADDRESS, que son dos comandos introducidos a modo de guía y que muestran información relativa a los distintos parámetros admitidos como datos y como direcciones. Esta información la detallamos a continuación:

    Data Parameters
    Un dato es un valor númerico en formato binario, octal, decimal o hexadecimal. El formato está especificado mediante los prefijos de caracteres:

    % (percent) -> número binario
    @ (commercial at) -> número octal
    & (ampersand) -> número hexadecimal
    $ (dollar) -> número decimal
    none (por defecto)-> número hexadecimal

    Address Parameters
    Los siguientes formatos de dirección serán aceptados:
    Address -> absoluto
    Address+offset -> absoluto con desplazamiento registro
    (A@) -> indirecto
    (A@,D@) -> indirecto con registro índice
    Address(A@) -> indirecto con desplazamiento
    Address(A@,D@) -> indirecto con registro índice y desplazamiento

LO1 [; [<options>]]
Almacena en memoria el archivo S-record cargado, donde <options> es una de las siguientes opciones:

- -C ignora el checksum durante la carga del programa
- X muestra los datos recibidos por consola\n\n")

MD <address> [<count>][; DI]
Muestra por consola <count> bytes de memoria desde <address>, donde:

- <address> es la dirección inicial del sector de memoria a mostrar.
- <count> es el nº de bytes (16 por defecto) a mostrar introducido en formato binario, octal, decimal o hexadecimal. Si no hay <count> específico se desensambla una sola instrucción.
- DI desensambla linea a linea el nº de bytes especificado

MM <address> [; DI]
Muestra / modifica el contenido de la memoria, donde

- <address> es la dirección de memoria a modificar
- DI activa el desensamblado en la visualización del contenido

Se muestra la dirección examinada y su actual contenido y activa una subrutina con la marca '?' en la que el siguiente subcomando es válido:

- <data> valor numérico a almacenar en el registro

NOBR [<address>] [<address>] ...
Elimina el breakpoint de <address> donde:

- <address> es la dirección de memoria en la que el programa parará. Usado sin parámetros borra todos los breakpoints existentes.

.PC [<address>]
Muestra / modifica el contenido del PC (Contador de Programa) donde:

- <address> es la dirección almacenada en el registro

.SR [<data>]
Muestra / modifica el contenido del Registro de Estado donde:

- <data> es la dirección almacenada en el registro

.SS [<data>]
Muestra / modifica el contenido del Puntero de Pila de Supervisor donde:

- <data> es la dirección almacenada en el registro

TR [<count>]
Ejecuta el número de instrucciones especificadas donde:

- <count> es el nº de intrucciones a tracear. Si no hay especificado, se tracea una instrucción

Una vez traceada, con un enter causamos un traceo más

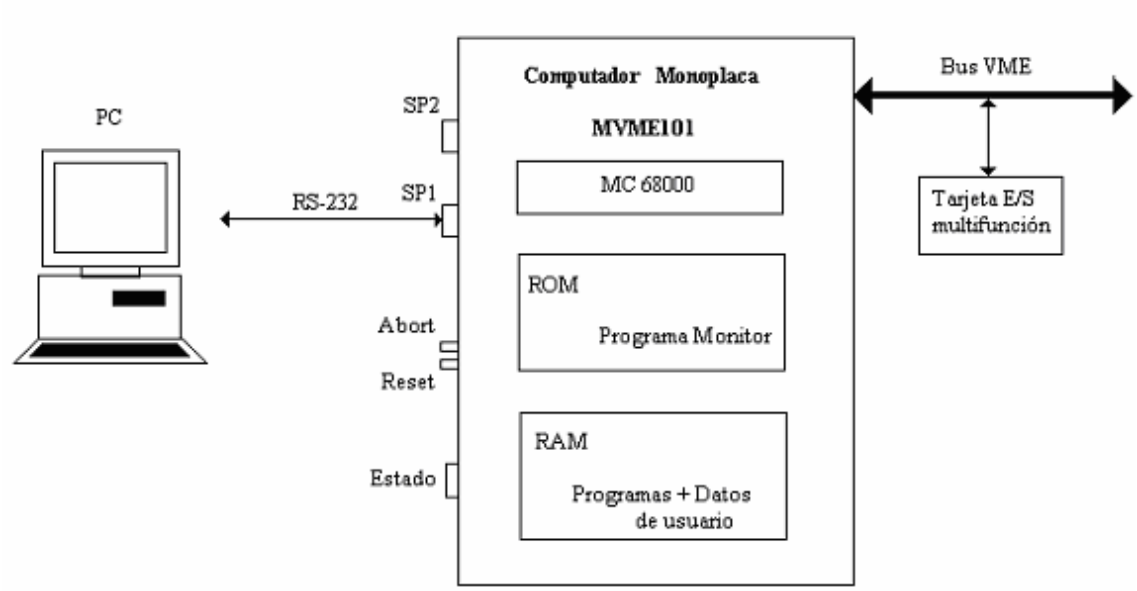
.US [<data>]
Muestra / modifica el contenido del Puntero de Pila de Usuario donde:

- <data> es la dirección almacenada en el registro

2.3 Interfaz de E/S

2.3.1 Descripción del puesto de trabajo

Antes de todo comenzaremos realizando una breve descripción de la organización física externa del sistema simulado. El proyecto realizado pretende ceñirse plenamente a esta organización, para facilitar más aún el conocimiento y manejo de la aplicación a todos aquellos usuarios ya familiarizados con el puesto de trabajo, con los cambios mínimos necesarios y la ventaja que supone el ser una aplicación software y no todo un equipo hardware.



El puesto de trabajo sobre el que se realizarán las prácticas está integrado por dos elementos básicos:

(a). Computador que se utiliza como sistema de desarrollo de las aplicaciones. El software usado:

- Ensamblador/Cargador cruzado para **MC68000**.
- Programa de emulación del terminal y/o transferencia de archivos.

(b). Equipo formado por un sistema microcomputador **MVME101** de Motorola utilizado como máquina objetivo. Características:

- Basado en **MC68000**, con 12K de **RAM** de usuario ampliables y salida bus **VME**.

- **Programa Monitor** en **EPROM**, para la carga y depuración de programas objeto.

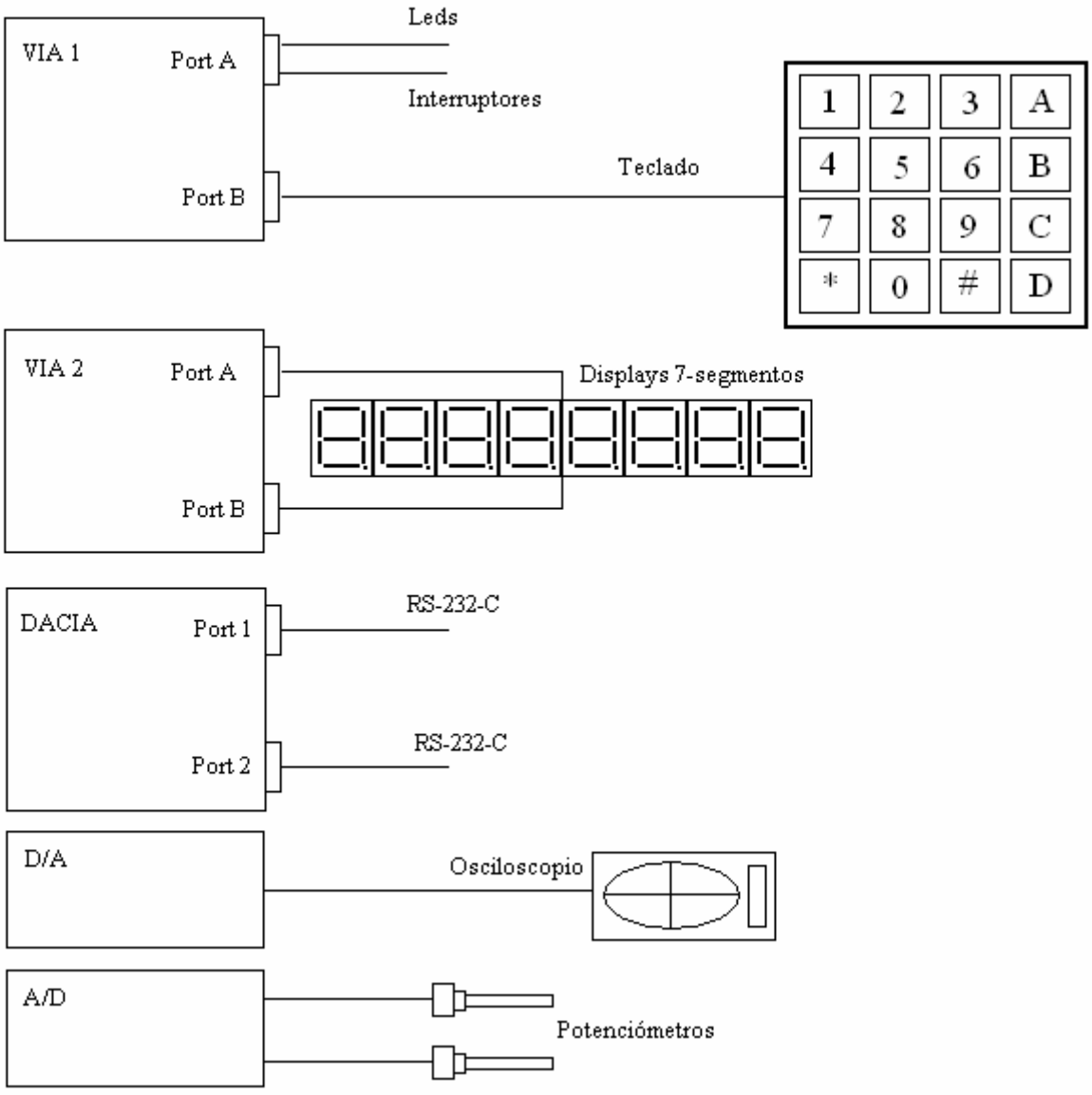
- Dos puertos serie, **SP1**(conectado al **PC**) y **SP2**, para comunicación con el exterior, un display indicador de estado, además de dos pulsadores **RESET** y **ABORT**.

- Tarjetas de interfaces de E/S conectada al microprocesador por medio del bus **VME**. Los interfaces son todos de 8 bits, e incluyen: VIAs, **DACIA**, conversores **A/D** y **D/A**.

- Tarjeta de periféricos conectada a la anterior, y accesible físicamente. La tarjeta contiene leds, interruptores, un banco de 8 displays de 7 segmentos, un teclado matricial 4X4, 2 puertos serie con drivers **RS-232C** (conectores DCE y DTE), 2 entradas analógicas conectadas a sendos potenciómetros, y 2 salidas analógicas para conexión a un osciloscopio.

2.3.2 Descripción de los periféricos

Vamos a realizar una descripción más detallada de los distintos periféricos y controladores incluidos en la tarjeta multifunción de E/S.



2.3.2.1 Descripción de la VIA .

El Circuito Integrado VIA (Versatile Interface Adapter) R6522 es un controlador de E/S paralela, con los siguientes elementos:

- Dos puertos de E/S paralelos de 8 líneas cada uno (registros ORA y ORB). Cada línea se puede programar independientemente para E ó S, actuando sobre los bits correspondientes de los registros DDRA ó DDRB.
- Dos pares de líneas (CA1, CA2, CB1, CB2), uno asociado a cada puerto, para examinar el estado de los periféricos y generar señales de control.
- Dos contadores/temporizadores de 16 bits (T1 y T2), que pueden utilizarse para generar retardos o pulsos y para detectar un número determinado de pulsos de entrada.
- Un registro de desplazamiento de 8 bits (SR), utilizable para E/S serie.
- Un registro de flags de 8 bits (IFR) que permite identificar la ocurrencia de determinados sucesos. Un registro de capacitación (IER) permite que la activación de un flag de IFR produzca una interrupción.
- Registros de control (ACR y PCR) para programar diferentes aspectos del funcionamiento de la VIA.

Los diferentes registros de cada una de las VIA's están proyectados en memoria. El acceso a estos se realiza por medio de operaciones de lectura y/o escritura en memoria. Cada uno de los registros tiene asignada una dirección en memoria, que son:

REGISTROS	**DIRECCIÓN**
ORB	EE0x01
ORA	EE0x03
DDRB	EE0x05
DDRA	EE0x07
T1-L	EE0x09
T1-H	EE0x0B
T2-L	EE0x11
T2-H	EE0x13
SR	EE0x15
ACR	EE0x17
PCR	EE0x19
IFR	EE0x1B
IER	EE0x1D

En donde x=0 para la VIA1 y x=2 para la VIA2.

2.3.2.2 Conexión de la VIA1.

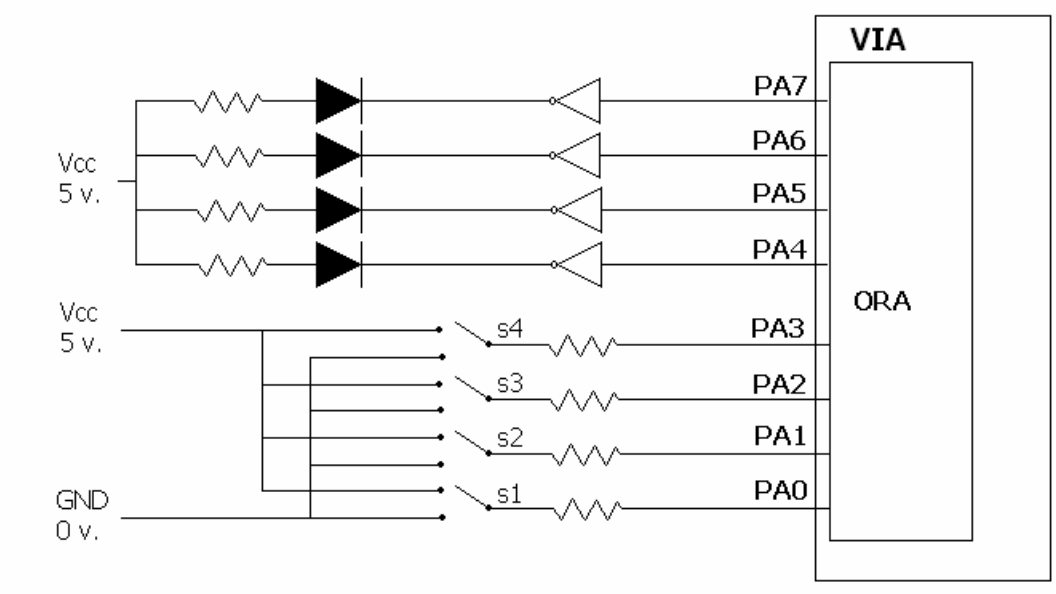
Las 4 líneas más significativas del puerto ORA de la VIA1 están conectadas a 4 leds, y las 4 menos significativas a 4 interruptores.

Como se ve en la figura, los interruptores están forzados a 0 voltios por medio de resistencias mientras están en estado “off”, mientras que en estado “on” pasan a 1 (5 voltios), los leds se iluminarán cuando se conduzca a través de los diodos (entren 0 voltios), esto será cuando las líneas de salida más significativas estén a 1 voltio.

El usuario será el encargado de programar las líneas de E y/o S del puerto mediante los registros DDRA o DDRB:

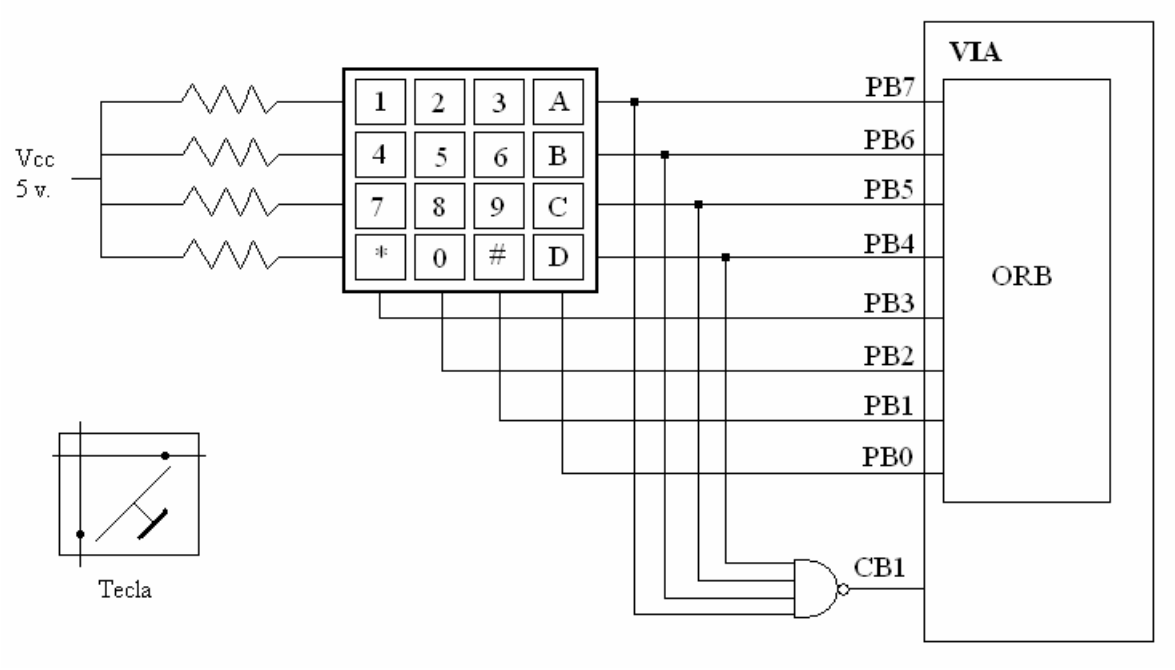
- Bit **n** de DDRx = 0 ➔ bit **n** de ORx está programado como entrada.
- Bit **n** de DDRx = 1 ➔ bit **n** de ORx está programado como salida.

Puerto A



Puerto B

El puerto B de la VIA 1 corresponde al teclado matricial 4x4 teclas, identificadas con 10 dígitos decimales, las letras A-D y los símbolos '*' y '#'



Las 4 columnas de la matriz están conectadas a las líneas menos significativas del puerto ORB, y las 4 filas a las más significativas. Estas últimas se mantienen a 1 continuo, y las columnas además entran a una puerta NAND cuya salida conecta a la línea CB1 de la VIA.

Procesamiento de la pulsación: el mecanismo de procesamiento de una pulsación de tecla es el siguiente:

(a). Detección de la pulsación: pasos:

- Se programan como salida las líneas menos significativas PB3-PB0.
- Programar la VIA mediante el registro PCR para que detecte transición por CB1.
- Poner a baja (valor '0') las líneas PB3-PB0 .

Al pulsar una tecla, las líneas correspondientes a las filas pasan a estar forzadas a '0', y la salida CB1 pasará de '0' (ninguna tecla pulsada) a '1' (alguna tecla pulsada).

Para ello hay que programar el registro PCR con el valor adecuado para que detecte transición y de que tipo en la línea CB1:

BIT PCR	Valor	Función
0	0	CA1 detecta transición negativa
0	1	CA1 detecta transición positiva
4	0	CB1 detecta transición negativa
4	1	CB1 detecta transición positiva

La detección de la transición programada, se indica en el registro IFR en el bit adecuado:

BIT IFR	Valor	Función
1	1	Ha ocurrido transición por CA1
1	0	No ha ocurrido transición por CA1
4	1	Ha ocurrido transición por CB1
4	0	No ha ocurrido transición por CA1

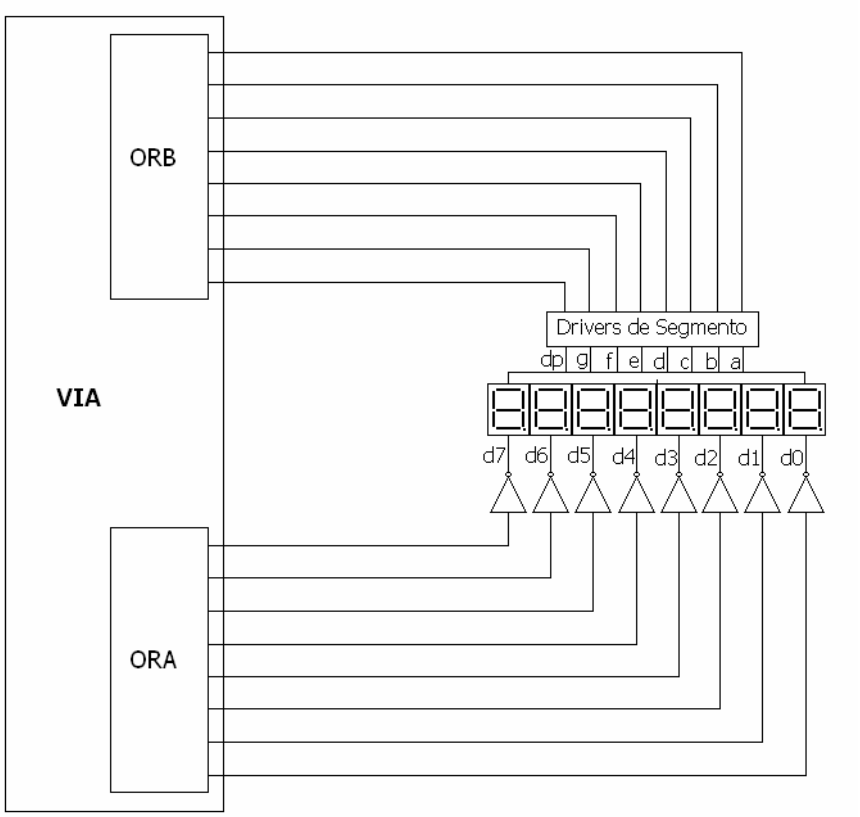
Para la detección del suceso de tecla pulsada o soltada, el programa debería evaluar de forma continua el estado del bit de IFR de la línea CB1.

(b). Identificación de la tecla pulsada: después de detectada la transición debe identificarse la tecla que ha sido pulsada.

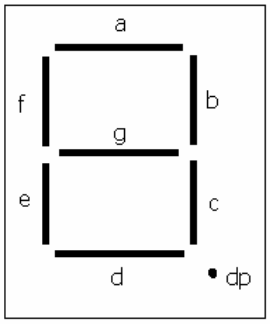
Antes de nada, hay que avisar que la transmisión no es instantánea, por lo que habrá que esperar un cierto tiempo para que la señal se estabilice en sus niveles de tensión. Igualmente ocurrirá en el momento de depresión de la tecla, la señal volverá a oscilar hasta que se estabilice. Por tanto en ambos casos antes de la identificación, se debe esperar un tiempo de 10ms para eliminar estos rebotes. Debido a que esta espera se programa así en el puesto de trabajo físico, en el simulador nos hemos visto obligados a también realizar esta “oscilación” por rebotes e identificar la tecla al azar si los intervalos de presión o depresión de las teclas son demasiado rápidos.

2.3.2.3 Conexión de la VIA2.

En la VIA2 ambos puertos, ORA y ORB, están conectados a los 8 displays. Los dos puertos se usan como salida por lo que deben programarse.
El puerto ORB se encarga de almacenar el código de 7-segmentos que se mostrará por el display. Mientras, el puerto ORA se encargará de seleccionar el display por el que se visualizará dicho código.



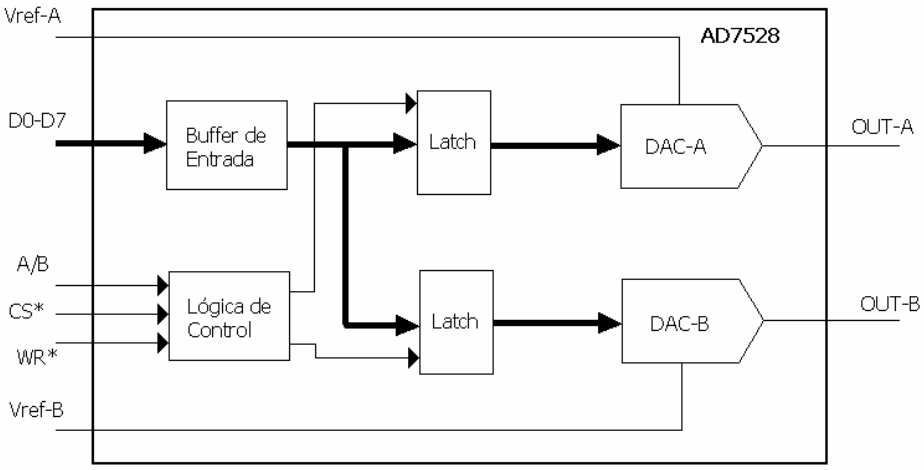
Cada línea de ORA selecciona un único display, mientras que cada línea de ORB se corresponde con un segmento del código a mostrar. El usuario simplemente deberá programar el display por el que desea visualizar, y el código a mostrar por dicho display.



2.3.2.4 Conversor D/A.

El CI AD7528 integra dos conversores digitales-analógicos, DAC-A y DAC-B, en un único chip. Se trata de conversores de 8 bits con salida de corriente, y cada uno con un registro interno que le permite la permanencia indefinida de la configuración de entrada.

Este conversor D/A lo utilizaremos para la conexión al osciloscopio, en donde mostraremos el contenido de los registros programados por el usuario.



El acceso se hace mientras un único conjunto de líneas de datos digitales D0-D7, conectadas a las líneas correspondientes del bus de datos del MC68000.
La selección del conversor al que se accede se hace mientras una línea específica.
Las direcciones que seleccionan a uno u otro registro son:

REGISTROS	**DIRECCIÓN**
DAC-A	EE0401
DAC-B	EE0403

La conexión de los conversores D/A al MVME101 se realiza directamente a través del bus. La carga del registro de entrada tiene lugar cuando se realiza una operación sobre la dirección asociada al conversor utilizado.

Las salidas están disponibles en los conectores del panel de E/S, junto con la entrada para la referencia de tierra analógica. El módulo de la salida es proporcional al valor binario de entrada interpretado como número sin signo, según se indica en la siguiente tabla:

ENTRADA DIGITAL								**Salida analógica Unipolar**
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**	**(OUT-A, OUT-B)**
1	1	1	1	1	1	1	1	+Vref * (256/256)
			...					...
1	0	0	0	0	0	0	1	+Vref * (129/256)
1	0	0	0	0	0	0	0	+Vref * (128/256)
0	1	1	1	1	1	1	1	+Vref * (127/256)
			...					...
0	0	0	0	0	0	0	0	+Vref * (0/256)

2. Modelo Java. Organización de paquetes

El proyecto está compuesto por tres grandes paquetes con los que simulamos fielmente todo el comportamiento del puesto de trabajo descrito anteriormente, tanto del Programa Monitor como de los periféricos.

El paquete **extras** servirá como paquete con métodos y parámetros usados en toda la aplicación, que sin embargo no se engloban en ningún otro paquete en concreto.

El paquete **motorola** representa la implementación del Motorota 68000, con todas las clases necesarias para la memoria y banco de registros, instrucciones, compilador del Programa Monitor, etc… Lo componen principalmente los siguientes paquetes:

1. ***Alu:***Unidad Aritmetico-Lógica del Motorola. Se encarga de ejecutar todas las operaciones necesarias en los programas.Tras realizar una de estas operaciones aritmeticológicas, se genera el vector de condiciones además del resultado.

2. ***Ejecutor:*** contiene los archivos necesarios para implementar el ejecutor de las instrucciones y el hilo de ejecución del programa.

3. ***Instrucciones:*** se encarga de reconocer e interpretar las instrucciones, leyendo el contenido binario de memoria y construyendo con esto la estructura de la instrucción para poder ejecutarla posteriormente.

4. ***Memoria:*** Memoria del Motorola. Esta definida como direccionable por Bytes con 12K de RAM de usuario y cumple todas las restricciones en cuanto accesos y modificaciones que vienen determinados en el manual oficial del Motorola 68K.

5. ***Registros:*** Banco de Registros del Motorola, con las implementaciones de los registros de direcciones, datos, estado y contador de programa.

El paquete **periféricos** contiene toda la implementación de las clases utilizadas para representar la Tarjeta Multifunción de E/S, la consola de interacción entre el usuario y el Programa Monitor, y el Osciloscopio. Este apartado está integrado por 3 paquetes principales:

1. ***Circuito VIA:*** contiene los archivos utilizados para implementar los controladores y periféricos de E/S de la tarjeta multifunción.
2. ***Consola:*** esta formado por los archivos que implementan la consola con el Programa Monitor con el que interactúa el usuario.
3. ***Osciloscopio:*** contiene los archivos para implementar el funcionamiento del osciloscopio.

3.1 Extras

El paquete de extras se utiliza a lo largo de la aplicación como un paquete auxiliar con métodos y clases necesarias, pero que no se integran en ningún otro paquete. Cualquier método para la conversión de valores binarios, complemento a 2, carga de archivos con filtro, etc… están disponibles en este paquete.

Extras →Binario

extras

Clase Binario

public class **Binario**

Clase Binario para tratar los vectores de bits y operar con ellos
Title: **BINARIO** Copyright (c) 2005

Detalle del constructor

Binario

`public Binario()`

Constructora de la clase **Binario**

Detalles de los métodos

compruebaBits

`public boolean compruebaBits(java.util.Vector cadena)`

Método que comprueba que las componentes de un vector son exclusivamente ceros y unos

Parameters: `cadena` - cadena a comprobar
Returns: indica si la cadena de bits es correcta o no

VectorBinarioAValor

`public static long VectorBinarioAValor(java.util.Vector v)`

Método que convierte un vector binario en su valor decimal. Recibirá vectores de longitud .B .W y .L correctos

Parameters: `v` - vector binario
Returns: valor decimal

VectorBinarioAValorSinSigno

`public static long` **`VectorBinarioAValorSinSigno`**`(java.util.Vector v)`

Método que convierte un vector binario en su valor decimal sin signo. Recibirá vectores de longitud .B .W y .L correctos

Parameters: `v` - vector en binario
Returns: valor decimal sin signo

ValorAVectorBinario

`public static java.util.Vector` **`ValorAVectorBinario`**`(long b, char tam)`

Método que convierte un valor decimal a un vector binario

Parameters: `b` - valor decimal
`tam` - tamaño del vector: Byte, Word, Longword
Returns: vector binario del valor decimal

---

getTamOperando

`public static int` **`getTamOperando`**`(char tam)`

Método que devuelve el tamaño del operando según sea .B, .W o .L (Byte=8 bits, Word=16 bits o Long=32 bits)

Parameters: `tam` - tamaño del operando
Returns: tamaño númerico del operando

StringBinarioAVectorBinario

`public static java.util.Vector`
`StringBinarioAVectorBinario``(java.lang.String valor, char tam)`

Método que convierte un valor en string a un vector binario

Parameters: `valor` - valor en string a convertir
`tam` - tamaño del vector: Byte, Word, Longword
Returns: vector binario del valor

VectorBinarioAString

`public static java.lang.String`
`VectorBinarioAString``(java.util.Vector valor, char tam)`

Método que convierte un vector binario a string

Parameters: `valor` - vector con el valor binario
`tam` - tamaño del vector: Byte, Word, Longword
Returns: string con el valor del vector

Extras → Comunes

extras

Clase Comunes

public class **Comunes**

Clase con operaciones comunes usadas en la aplicación
Title: **COMUNES** Copyright (c) 2005

Detalle del constructor

Comunes

```
public Comunes()
```

Constructora de la clase **Comunes**

Detalles de los métodos

calculaValorNumerico

```
public static long calculaValorNumerico(java.lang.String datos)
```

Función que calcula el valor numérico decimal de un dato representado en Binario, Octal o Hexadecimal

Parameters: `datos` - datos cuyo valor númerico calculamos

---

hexadecimalToDecimal

```
public static long hexadecimalToDecimal(java.lang.String cadenaHex)
```

Método por el que a partir de una cadena en hexadecimal se representa una cadena en decimal

Parameters: `cadenaHex` - cadena en hexadecimal
Returns: cadena en decimal

---

cualquierBaseAdecimal

```
public static long cualquierBaseAdecimal(java.lang.String cadena,
                                         int base)
```

Método que devuelve el valor decimal de un string en base indicada

Parameters: `cadena` - string con el número a calcular
`base` - base en la que está representado el número de la cadena
Returns: cadena en decimal

decimalToBinario

```
public static java.lang.String decimalToBinario(long valor)
```

Método por el que a partir de un valor que representa un valor decimal se devuelve en un string su valor en binario

Parameters: `valor` - valor decimal
Returns: cadena en binario

---

binarioAHexadecimal

```
public static java.lang.String
binarioAHexadecimal(java.lang.String valorBinario)
```

Método que convierte un valor binario representado en su valor en hexadecimal

Parameters: `valorBinario` - cadena en binario
Returns: cadena en hexadecimal

---

par

```
public static boolean par(long direccion)
```

Método que devuelve si una dirección es par

Parameters: `direccion` - dirección a evaluar
Returns: devuelve si la dirección es par

---

impar

```
public static boolean impar(long direccion)
```

Método que devuelve si una dirección es impar

Parameters: `direccion` - dirección a evaluar
Returns: devuelve si la dirección es impar

---

completaCon

```
public static java.lang.String completaCon(java.lang.String cadena,
                                           char caracter,int cantidad)
```

Completa un String con tantas instancias de un mismo caracter como le indiquemos

Parameters: `cadena` - cadena a completar
`caracter` - caracter de relleno para la cadena
`cantidad` - número de repeticiones del caracter de relleno
Returns: cadena final

invertir

`public static java.util.Vector` **`invertir`**`(java.util.Vector op)`
Método que invierte un Vector
Parameters: op - vector a invertir
Returns: vector invertido

Extras → Constantes

extras

Clase Constantes

public class **Constantes**
Clase para representar constantes .B, .W y .L
Title: **CONSTANTES** Copyright (c) 2005

Detalle del campo

bit

`public static final char` **`bit`**
Caracter bit (1 bit)

Byte

`public static final char` **`Byte`**
Caracter Byte (8 bits)

Word

`public static final char` **`Word`**
Caracter Word (16 bits)

Long

`public static final char` **`Long`**
Caracter Long (32 bits)

Especial

`public static final char` **`Especial`**
Caracter especial

CaracterVacio

```
public static final char CaracterVacio
```
Caracter vacio

Extras → FiltroURL

extras

Clase FiltroURL

public class **FiltroURL**
extends javax.swing.filechooser.FileFilter
Clase Filtro usado a la hora de cargar archivos .HEX y filtrarlos
Title: **FILTRO URL** Copyright (c) 2005

Detalle del constructor

FiltroURL

```
public FiltroURL()
```
Constructora de la clase **FiltroURL** por defecto

FiltroURL

```
public FiltroURL(java.lang.String filtro)
```
Constructora de la clase a traves de una cadena con el filtro a aplicar
Parameters: `filtro` - valor del filtro

Detalles de los métodos

getDescription

```
public java.lang.String getDescription()
```
Función que nos dice una breve descripción del tipo de los ficheros que filtramos
Returns: String con la descripción

---

accept

```
public boolean accept(java.io.File f)
```
Redefinición de la función accept, para poder filtrar los archivos de tipo "*.hex"
Parameters: `f` - nombre del fichero que queremos cargar
Returns: booleano que indica si el fichero f cumple nuestro filtro y por tanto podemos cargarlo
See Also: javax.swing.filechooser.FileFilter#accept(File f)

Extras → Par

extras

Clase Par

public class **Par**

Clase para representar un par de objetos de la forma (Par1;Par2)
Title: **PAR** Copyright (c) 2005

Detalle del constructor

Par

```
public Par(java.lang.Object p,java.lang.Object s)
```

Constructora de la clase **Par**

Parameters: p - primer elemento de la pareja
s - segundo elemento de la pareja

Detalles de los métodos

getPrimero

```
public java.lang.Object getPrimero()
```

Método accesor para el primer elemento

Returns: primer elemento

---

getSegundo

```
public java.lang.Object getSegundo()
```

Método accesor para el segundo elemento

Returns: segundo elemento .

---

setPrimero

```
public void setPrimero(java.lang.Object p)
```

Método modificador del primer elemento

Parameters: p - nuevo elemento

---

setSegundo

```
public void setSegundo(java.lang.Object s)
```

Método modificador del segundo elemento

Parameters: s - nuevo elemento

3.2 Motorola

Motorola →ALU

Este paquete contiene únicamente a la clase ALU, que es la encargada de simular el comportamiento de una unidad aritmético-lógica real y semejante a la implementada en el Motorola 68000.La idea general es que realiza las operaciones a partir de uno o dos operandos binarios, con la precisión que se le solicite (Byte, Word y Long Word) devolviendo el resultado en formato binario y generando asi mismo un vector de condiciones semejante a los bits generados a la ALU real y referentes al acarrero, el desbordamiento, el bit de cero y el bit de negativo.

Las instrucciones que realiza la ALU son todas las instrucciones enteras del repertorio (Las de punto flotante no se tratan en la asignatura para la que se ha diseñado este simulador, asi que era innecesario implementarlas).Pasamos a ver el java doc generado para esta clase, donde veremos las instrucciones implementadas y otros metodos:

motorola.alu

ClaseAlu

public class **Alu**

Clase que implementa a una unidad aritmetico-lógica ALU
Title: **ALU** Copyright (c) 2005

Detalle del constructor

Alu

`public Alu()`

Constructora de la clase **Alu**

Detalles de los métodos

AluAdd

`public Par AluAdd(Vector op1,Vector op2)`

Operación de suma de dos vectores binarios

Parameters: `op1` - primer operando
`op2` - segundo operando

Returns: valor par con (resultado, condiciones)

AluAddU

`public Par` **`AluAddU`**`(Vector op1, Vector op2)`

Operación de suma de dos vectores binarios sin signo

Parameters: `op1` - primer operando
`op2` - segundo operando

Returns: valor par con (resultado, condiciones)

AluSub

`public Par` **`AluSub`**`(Vector op1, Vector op2)`

Operación de resta de dos vectores binarios

Parameters: `op1` - operando minuendo
`op2` - operando sustraendo

Returns: valor par con (resultado, condiciones)

AluSubU

`public Par` **`AluSubU`**`(Vector op1, Vector op2)`

Operación de resta sin signo de dos vectores binarios

Parameters: `op1` - operando minuendo
`op2` - operando sustraendo

Returns: valor par con (resultado, condiciones)

AluMul

`public Par` **`AluMul`**`(Vector op1, Vector op2)`

Operación de multiplicación de dos vectores binarios

Parameters: `op1` - primer operando
`op2` - segundo operando

Returns: valor par con (resultado, condiciones)

AluMulU

`public Par` **`AluMulU`**`(Vector op1, Vector op2)`

Operación de multiplicación sin signo de dos vectores binarios

Parameters: `op1` - primer operando
`op2` - segundo operando

Returns: valor par con (resultado, condiciones)

AluDiv

`public Par` **`AluDiv`**`(Vector op1, Vector op2)`

Operación de división de dos vectores binarios

Parameters: `op1` - operando dividendo
`op2` - operando divisor

Returns: valor par con (resultado, condiciones)

AluDivU

`public Par` **`AluDivU`**`(Vector op1, Vector op2)`

Operación de división sin signo de dos vectores binarios

Parameters: `op1` - operando dividendo
`op2` - operando divisor

Returns: valor par con (resultado, condiciones)

AluDespLogIzq

`public Par` **`AluDespLogIzq`**`(Vector op1, Vector desplaz)`

Operación de desplazamiento lógico a la izquierda de un vector binario

Parameters: `op1` - primer operando
`desplaz` - tamaño del desplazamiento

Returns: valor par con (resultado, condiciones)

AluDespLogDer

`public Par` **`AluDespLogDer`**`(Vector op1, Vector desplaz)`

Operación de desplazamiento lógico a la derecha de un vector binario

Parameters: `op1` - primer operando
`desplaz` - tamaño del desplazamiento

Returns: valor par con (resultado, condiciones)

AluDespAritIzq

`public Par` **`AluDespAritIzq`**`(Vector op1, Vector desplaz)`

Operación de desplazamiento aritmético a la derecha de un vector binario.
Identico al desplazamiento lógico a izq. menos en las condiciones generadas

Parameters: `op1` - primer operando
`desplaz` - tamaño del desplazamiento

Returns: valor par con (resultado, condiciones)

AluDespAritDer

`public Par` **`AluDespAritDer`**`(Vector op1, Vector desplaz)`

Operación de desplazamiento aritmético a la derecha de un vector binario. Se diferencia del lógico en que extiende el signo al desplazar

Parameters: `op1` - primer operando
`desplaz` - tamaño del desplazamiento

Returns: valor par con (resultado, condiciones)

AluROL

`public Par` **`AluROL`**`(Vector op1, Vector desplaz)`

Operación de rotación a la izquierda de un vector binario

Parameters: `op1` - operando
`tamaño` - de la rotación

Returns: valor par con (resultado, condiciones)

AluROR

`public Par` **`AluROR`**`(Vector op1, Vector desplaz)`

Operación de rotación a la derecha de un vector binario

Parameters: `op1` - operando
`tamaño` - de la rotación

Returns: valor par con (resultado, condiciones)

AluNOT

`public Par` **`AluNOT`**`(Vector op1)`

Operación de NOT de un vector binario

Parameters: `op1` - operando

Returns: valor par con (resultado, condiciones)

AluAND

`public Par` **`AluAND`**`(Vector op1, Vector op2)`

Operación de multiplicación binario AND de dos vectores binarios

Parameters: `op1` - primer operando
`op2` - segundo operando

Returns: valor par con (resultado, condiciones)

AluOR

```
public Par AluOR(Vector op1, Vector op2)
```

Operación de OR de dos vectores binarios

Parameters: `op1` - primer operando
`op2` - segundo operando

Returns: valor par con (resultado, condiciones)

AluXOR

```
public Par AluXOR(Vector op1, Vector op2)
```

Operación de XOR de dos vectores binarios

Parameters: `op1` - primer operando
`op2` - segundo operando

Returns: valor par con (resultado, condiciones)

AluCLEAR

```
public Par AluCLEAR(Vector op1)
```

Operación de borrado de un vector binario

Parameters: `op1` - primer operando

Returns: valor par con (resultado, condiciones)

AluNeg

```
public Par AluNeg(Vector op1)
```

Operación de complemento a 2 del vector binario entrante

Parameters: `op1` - operando

Returns: valor par con (resultado, condiciones)

AluExt

```
public Par AluExt(Vector op1,
                  char tam)
```

Operación de extensión del signo de un vector al binario a la longitud solicitada

Parameters: `op1` - operando
`tam` - tamaño de la extensión

Returns: valor par con (resultado, condiciones)

AluMove

```
public Par AluMove(Vector op1)
```

Operación de move, se hace para generar las condiciones

Parameters: `op1` - operando

Returns: valor par con (resultado, condiciones)

generaCero

`private int` **`generaCero`**`(Vector v)`

Método que genera la condición de Zero de un operando

Parameters: `v` - operando
Returns: valor de la condición

---

generaNegativo

`private int` **`generaNegativo`**`(Vector v)`

Método que genera la condición de Negativo de un operando

Parameters: `v` - operando
Returns: valor de la condición

---

generaCarry

`private int` **`generaCarry`**`(long resultado, char tam)`

Método que genera la condición de Carry de un operando

Parameters: `resultado` - operando
`tam` - tamaño maximo del operando
Returns: valor de la condición

---

generaOverflow

`private static int` **`generaOverflow`**`(long resultado, char tam)`

Método que genera la condición de Overflow de un operando

Parameters: `resultado` - operando
`tam` - tamaño maximo del operando
Returns: valor de la condición

Motorola →Instrucciones

Este paquete esta compuesto por dos clase, la clase Instrucción, que define el formato de las instrucciones, sus formatos de impresión por pantalla.. y la clase InterpreteInstrucciones, que se encarga de decodificar y tratar los 1´s y 0´s almacenados en memoria pera convertirlos en instrucciones en ensamblador.

El Java doc de la clase Instrucciones es:

Motorola →Instrucciones →Instrucion

motorola.instrucciones

Clase Instruccion

public class **Instruccion**

Clase para representar las instrucciones ensamblador una vez decodificada de la memoria
Title: **INSTRUCCIÓN** Copyright (c) 2005

Detalle del constructor

Instruccion

```
public Instruccion(java.lang.String tipo, int rf, int inmed,
                   java.lang.String direc,java.lang.String direcf,
                   char tamaño, int rd,java.lang.String destSal,
                   java.lang.String absFuente,
                   java.lang.String absDestino,int despPC,int despf,
                   int despd,int rdespf,int rdespd,char szrdespf,
                   char szrdespd)
```

Constructora de la clase **Instrucción**

Parameters:

`tipo` - código de la instrucción
`rf` - registro fuente
`inmed` - operando inmediato
`direc` - modo direccionamiento destino
`direcf` - modo direccionamiento fuente
`tamaño` - modo de operación de la instrucción
`rd` - registro destino
`destSal` - destino del salto
`absFuente` - absoluto de la fuente
`absDestino` - absoluto del destino
`despPC` - desplazamiento del PC
`despf` - desplazamiento de la fuente
`despd` - desplazamiento del destino
`rdespf` - registro desplazamiento de la fuente
`rdespd` - registro desplazamiento del destino
`szrdespf` - tamaño del registro desplazamiento de la fuente
`szrdespd` - tamaño del registro desplazamiento del destino

Detalles de los métodos

EscribeInstruccion

```
public java.lang.String EscribeInstruccion()
```
Método que escribe la instrucción desensamblada para mostrar por consola
Returns: instrucción desensamblada

RepresentaDireccionEfectiva

```
public java.lang.String RepresentaDireccionEfectiva(boolean esDestino)
```
Método que representa en formato legible la dirección efectiva del operando
Parameters: `esDestino` - indica si es destino o no el operando
Returns: representación de la dirección efectiva

Motorola →Instrucciones → InterpreteInstruciones

motorola.instrucciones

Clase InterpreteInstrucciones

public class **InterpreteInstrucciones**
Clase que implementa al interprete de instrucciones ensamblador a traves del contenido en bits de la memoria
Title: **INTERPRETE INSTRUCCIONES** Copyright (c) 2005

Detalle del constructor

InterpreteInstrucciones

```
public InterpreteInstrucciones(motorola.memoria.Memoria mem,
                               motorola.registros.BancoRegistros b)
```
Constructora de la clase <B< de>
Parameters: `mem` - Memoria
`b` - Banco de Registros

Detalles de los métodos

DescodificaInstruccion

```
public motorola.instrucciones.Instruccion
DescodificaInstruccion(long dir)
```
Método que lee el código de la memoria y saca el tipo de instrucción
Parameters: `dir` - direccion de la instrucción que queremos analizar
Returns: instrucción descodificada

DefineDC

`private motorola.instrucciones.Instruccion` **`DefineDC`**`()`

Método que en caso de una deteccion erronea pone por defecto una DC restableciendo previamente todos los registros tocados

Returns: instrucción DC por defecto

---

sacaSize

`private char` **`sacaSize`**`(java.lang.String cadenabits)`

Método que determina la longitud de la operación con respecto a los bits correspondientes x00 -> .B x01 -> .W x10 -> .L codificacion en la pagina 4-5 si el tamaño es de solo dos bits esto tambien vale 00, 01, 11

Parameters: `cadenabits` - cadena de bits

Returns: char con el tamaño de la operación (.B, .W, .L)

---

sacaSizeMove

`private char` **`sacaSizeMove`**`(java.lang.String cadenabits)`

Método que determina la longitud de la operación MOVE con respecto a los bits correspondientes

Parameters: `cadenabits` - cadena de bits

Returns: char con el tamaño del MOVE (.B, .W, .L)

---

sacaModoDireccionamiento

`private java.lang.String`
`sacaModoDireccionamiento``(java.lang.String cadenabits,`
`int reg,boolean esFuente)`

Método que determina el modo de direccionamiento de los operandos de una instrucción

Parameters: `cadenabits` - cadena de bits de la instrucción
`reg` - registro
`esFuente` - indica si el operando es fuente o destino

Returns: cadena con el modo de direccionamiento (directo, indirecto, absoluto,...)

El listado de instrucciones implementadas y que el interprete de instrucciones es capaz de interpretar y posteriormente de crear una estructura que las defina es:

ADD, ADDA, ADDI, DBRA, DBSR, DBHI, DBLS, DBCC(HI), DBCC(LO), DBNE, DBEQ, DIC, DBVS, DBPL, DBMI, DBGE, DBLT, DBGT, DBLE, SUBQ, ADDQ, AND, ANDI, ASL, ASR, ASL, ASR, BRA, BSR, BHI, BLS, BCC(HI), BCC(LO), BNE, BEQ, BVC, BVS, BPL, BMI, BGE, BLT, BGT, BLE, MOVEP, BCHG, BCLR, BTST, BSET, CHK, CLR, ILLEGAL, JMP, JSR, EXT, EXTB, LEA, CMPM, CMP, CMPA, CMPI, DIVS, DIVSL, DIVU, DIVUL, EOR, EORI, EXG, LSL, LSR, LSL, LSR, MOVEA, MOVE, MOVE from CCR, MOVE to CCR, MOVE from SR, MOVEQ, MULS, MULU, NBCD, NEG, NEGX, NOP, NOT, OR, ORI, PEA, ROL, ROR, ROXL, ROXR, RTM, RTR, RTS, SUB, SUBA, SUBI, SUBX, SWAP, TAS, TRAP, TRAPV, TST.

Motorola → Compilador → Ejecutor

public class **Ejecutor**

Clase que implementa al ejecutor de las instrucciones ensamblador del Motorola
Title: **EJECUTOR**

Detalles del constructor

Ejecutor

```
public Ejecutor(motorola.alu.Alu a,
                motorola.registros.BancoRegistros b,
                motorola.memoria.Memoria m,
                motorola.compilador.HiloEjecucion h,
                perifericos.consola.MVME101 mvme,
                java.util.Hashtable breakpoints,long fin)
```

Constructora genérica de la clase **Ejecutor**

Parameters: `a` - Alu
`b` - Banco de Registros
`m` - Memoria
`h` - Hilo de Ejecucion
`mvme` - MVME
`breakpoints` - Tabla de Breakpoints
`fin` - direccion de finalización

Ejecutor

```
public Ejecutor(motorola.alu.Alu a,
                motorola.registros.BancoRegistros b,
                motorola.memoria.Memoria m)
```

Constructora de la clase **Ejecutor** (com TR), que inicializa todos sus valores

Parameters: `a` - Alu
`b` - Banco de Registros
`m` - Memoria

Detalles de los métodos

getBanco

```
public motorola.registros.BancoRegistros getBanco()
```

Método accesor al Banco de Registros

Returns: Banco de Registros

---

getMVME

```
public perifericos.consola.MVME101 getMVME()
```

Método accesor al MVME

Returns: MVME

calcularOperandoFuente

```
public java.util.Vector
calcularOperandoFuente(motorola.instrucciones.Instruccion ins,
                                                    java.lang.Object dir)
```

Método que obtiene el operando fuente de una instrucción buscado según su modo de direccionamiento

Parameters: `i` - instrucción

Returns: Operando fuente hallado según su modo de direccionamiento

---

calcularOperandoDestino

```
public java.util.Vector
calcularOperandoDestino(motorola.instrucciones.Instruccion ins,
                                                     java.lang.Object dir)
```

Método que obtiene el operando destino de una instrucción buscado según su modo de direccionamiento

Parameters: `i` - instrucción

Returns: Operando destino hallado según su modo de direccionamiento

calculaOperando

```
private java.util.Vector
calculaOperando(java.lang.Object dirEfectiva,
java.lang.String modoDireccionamiento, char longitud, int reg,
int desplaz, int regDesplaz,char tamRegDesp, int inmediato,
       java.lang.String absoluto)
```

Método para obtener dado un vector con el indirecto a memoria, el operando de memoria y su direccion

Parameters: `tam` - tamaño del operando(Constantes.Byte, Word, Long)
`vecDir` - vector con el indirecto a memoria (contenido de An)

Returns: Operando de memoria y su dirección

---

calculaDireccionEfectiva

```
public java.lang.Object
calculaDireccionEfectiva(java.lang.String modoDireccionamiento,
               char modoOperacion, int reg,int desp, int despReg,
               char tamRegDesp, int inmediato,
               java.lang.String absoluto)
```

Método que calcula la dirección efectiva de un operando según los parámetros

Parameters: `modoDireccionamiento` - modo de direccionamiento
`modoOperacion` - modo de operacion (Byte, Word, Long)
`reg` - numero de registro
`desp` - desplazamiento
`despReg` - desplazamiento en registro

Returns: Objeto con el valor apuntado por la dirección efectiva

calculaDireccionEfectivaDestino

```
private java.lang.Object
calculaDireccionEfectivaDestino(motorola.instrucciones.Instruccion i)
```

Método que calcula la dirección efectiva del destino

Parameters: `i` - instrucción

Returns: Objeto con el contenido de la dirección efectiva del destino

---

calculaDireccionEfectivaFuente

```
private java.lang.Object
calculaDireccionEfectivaFuente(motorola.instrucciones.Instruccion i)
```

Método que calcula la dirección efectiva del destino

Parameters: `i` - instrucción

Returns: Objeto con el contenido de la dirección efectiva de la fuente

almacenaResultado

```
private void almacenaResultado(extras.Par p,
                               java.lang.Object dirDestino, char modo)
```

Función que almacena el resultado de una operación en registro o en memoria según venga especificado

Parameters: `p` - Par con (resultado, condiciones)
`dirDestino` - dirección destino
`modo` - modo de operación (Byte, Word, Long)

---

almacenaBitResultado

```
private void almacenaBitResultado(java.lang.Object dirDestino,
                                  int numBit,int bit)
```

Función que almacena el resultado de una operación en registro o en memoria según venga especificado

Parameters: `dirDestino` - dirección destino
`numBit` - numero de bit a modificar
`bit` - valor nuevo del bit

---

actualizaCondiciones

```
private void actualizaCondiciones(extras.Par p)
```

Función que actualiza el vector de condiciones

Parameters: `p` - vector de condiciones nuevo

opADD

`private boolean` **`opADD`**`(java.lang.String op)`

Método que indica si una operación es de tipo ADD

Parameters:`op` - tipo de operación

Returns: Booleano que indica si la operación es del tipo indicado

---

opAND

`private boolean` **`opAND`**`(java.lang.String op)`

Método que indica si una operación es de tipo AND

Parameters:`op` - tipo de operación

Returns: Booleano que indica si la operación es del tipo indicado

opSUB

`private boolean` **`opSUB`**`(java.lang.String op)`

Método que indica si una operación es de tipo SUB

Parameters: `op` - tipo de operación

Returns: Booleano que indica si la operación es del tipo indicado

---

opOR

`private boolean` **`opOR`**`(java.lang.String op)`

Método que indica si una operación es de tipo OR

Parameters: `op` - tipo de operación

Returns: Booleano que indica si la operación es del tipo indicado

---

opCMP

`private boolean` **`opCMP`**`(java.lang.String op)`

Método que indica si una operación es de tipo CMP

Parameters: `op` - tipo de operación

Returns: Booleano que indica si la operación es del tipo indicado

---

opMOVE

`private boolean` **`opMOVE`**`(java.lang.String op)`

Método que indica si una operación es de tipo MOVE

Parameters: `op` - tipo de operación

Returns: Booleano que indica si la operación es del tipo indicado

opNEG

`private boolean` **`opNEG`**`(java.lang.String op)`

Método que indica si una operación es de tipo NEG

Parameters: `op` - tipo de operación
Returns: Booleano que indica si la operación es del tipo indicado

---

opXOR

`private boolean` **`opXOR`**`(java.lang.String op)`

Método que indica si una operación es de tipo XOR

Parameters: `op` - tipo de operación
Returns: Booleano que indica si la operación es del tipo indicado

---

incrementaPC

`private void` **`incrementaPC`**`(long longitud)`

Método que incrementa el valor del PC

Parameters: `longitud` - longitud a incrementar

---

modificaPC

`private void` **`modificaPC`**`(java.lang.String dirSalto)`

Método que modifica el PC con la dirección nueva

Parameters: `dirSalto` - dirección con la cual se va a modificar el PC

---

ejecutaInstruccion

`public void` **`ejecutaInstruccion`**`()`

Método principal que ejecuta la instrucción apuntada por el PC

Motorola → Compilador → HiloEjecución

motorola.compilador

Clase HiloEjecucion

public class **HiloEjecucion**
implements java.lang.Runnable

Clase que implementa el hilo de ejecución del ensamblador
Title: **HILO EJECUCIÓN** Copyright (c) 2005

Detalle del constructor

HiloEjecucion

`public` **`HiloEjecucion`**`()`

Constructora de la clase **Hilo de Ejecución** que crea el hilo y le concede prioridad mínima.

Detalles de los métodos

setEjecutor

`public void` **`setEjecutor`**`(motorola.compilador.Ejecutor e)`

Método que actualiza el Ejecutor del hilo

Parameters: `e` - ejecutor

---

setBr

`public void` **`setBr`**`(java.util.Hashtable breakpoints)`

Método que actualiza la presencia de breakpoints

Parameters: `breakpoints` - tabla de breakpoints

---

setDirFin

`public void` **`setDirFin`**`(long dir)`

Método que actualiza la dirección de finalización

Parameters: `dir` - dirección de finalización

---

getHR

`public java.lang.Thread` **`getHR`**`()`

Método accesor del Thread del hilo

Returns: Thread del hilo

---

run

`public void` **`run`**`()`

Método run para el hilo de ejecución. En caso de que no halla ningún breakpoint con valor ‘0’ en la dirección actual de ejecución, se disminuye su acumulador y se ejecuta la instrucción correspondiente, sino se para la ejecución en la dirección con breakpoint de valor ‘0’, se muestra el estado y se detiene el hilo.

hayBreakpoint

`private boolean` **`hayBreakpoint`**`()`

Método que indica si hay breakpoint en una dirección con acumulador a 0

Returns: indica si hay breakpoint

---

decrementaBreakpoint

`private void` **`decrementaBreakpoint`**`()`

Método que decrementa el contador del breakpoint

Returns:

Motorola → Memoria → Memoria

public class **Memoria**

Clase que implementa la Memoria del Motorola68000. Memoria direccionable por bytes y estructurada por bloques en vectores. Cada posición del vector es un String de 16 posiciones. Las 8 más significativas corresponden al primer byte accesible Las 8 menos significativas corresponden al segundo byte accesible Title: **MEMORIA**

Detalle del tamaño de los bloques de memoria

Bloques de direcciones de memoria del 68000 y sus tamaños en orden desde 000000 hasta FFFFFF

tamExcepciones

`private static final int` **`tamExcepciones`** `= 1024`

Posiciones 000000 - 0003FF de memoria

---

tamRAM_monitor

`private static final int` **`tamRAM_monitor`** `= 7168`

Posiciones 000400 - 001FFF de memoria

---

tamRAM_usuario

`private static final int` **`tamRAM_usuario`** `= 4096`

Posiciones 002000 - 002FFF de memoria

---

tamNoUsado1

`private static final int` **`tamNoUsado1`** `= 15585281`

Posiciones 003000 - EE0000 de memoria

tamPlacaMultifunciónES

```
private static final int tamPlacaMultifuncionES = 2047
        Posiciones EE0001 - EE07FF de memoria
```

---

tamNoUsado2

```
private static final int tamNoUsado2 = 129024
        Posiciones EE0800 - EFFFFF de memoria
```

---

tamProgramaMonitor

```
private static final int tamProgramaMonitor = 32768
        Posiciones F00000 - F07FFF de memoria
```

---

tamNoUsado3

```
private static final int tamNoUsado3 = 1015808
        Posiciones F08000 - FFFFFF de memoria
```

Direcciones de comienzo de los bloques de memoria

dirExcepciones

```
private static final int dirExcepciones = 0
```
000000 -> 0

---

dirRAM_monitor

```
private static final int dirRAM_monitor = 1024
        000400 -> 1024
```

---

dirRAM_usuario

```
private static final int dirRAM_usuario = 8192
        002000 -> 8192
```

---

dirNoUsado1

```
private static final int dirNoUsado1 = 12288
        003000 -> 12288
```

dirPlacaMultifuncionES

```
private static final int dirPlacaMultifuncionES = 15597569
        EE0001 -> 15597569
```

---

dirNoUsado2

```
private static final int dirNoUsado2 = 15599616
        EE0800 -> 15599616
```

---

dirProgramaMonitor

```
private static final int dirProgramaMonitor = 15728640
        F00000 -> 15728640
```

---

dirNoUsado3

```
private static final int dirNoUsado3 = 15761408
        F08000 -> 15761408
```

---

dirUltima

```
private static final int dirUltima = 15761408
        FFFFFF -> 16777215
```

Direcciones de los puertos señalados en las VIA

via1ORB

```
private static final String via1ORB = "EE0001"
```

---

via1ORA

```
private static final String via1ORA = "EE0003"
```

---

via1DDRB

```
private static final String via1DDRB = "EE0005"
```

via1DDRA

```
private static final String via1DDRA = "EE0007"
```

via1T1L

```
private static final String via1T1L = "EE0009"
```

---

via1T1H

```
private static final String via1T1H = "EE000B"
```

---

via1T2L

```
private static final String via1T2L = "EE0011"
```

---

via1T2H

```
private static final String via1T2L = "EE0013"
```

---

via1SR

```
private static final String via1SR = "EE0015"
```

---

via1ACR

```
private static final String via1ACR = "EE0017"
```

---

via1PCR

```
private static final String via1PCR = "EE0019"
```

---

via1IFR

```
private static final String via1IFR = "EE001B"
```

---

via1IER

```
private static final String via1IER = "EE001D"
```

---

via2ORB

```
private static final String via2ORB = "EE0201"
```

via2ORA

```
private static final String via2ORA = "EE0203"
```

---

via2DDRB

```
private static final String via2DDRB = "EE0205"
```

---

via2DDRA

```
private static final String via2DDRA = "EE0207"
```

---

via2T1L

```
private static final String via2T1L = "EE0209"
```

---

via2T1H

```
private static final String via2T1H = "EE020B"
```

---

via2T2L

```
private static final String via2T2L = "EE0211"
```

---

via2T2H

```
private static final String via2T2L = "EE0213"
```

---

via2SR

```
private static final String via2SR = "EE0215"
```

---

via2ACR

```
private static final String via2ACR = "EE0217"
```

---

via2PCR

```
private static final String via2PCR = "EE0219"
```

via2IFR

```
private static final String via2IFR = "EE021B"
```

---

via2IER

```
private static final String via2IER = "EE021D"
```

Detalle del constructor

Memoria

```
public Memoria()
```

Constructora de la clase **Memoria**

Detalles de los métodos

get

```
public java.lang.String get(long direccion,char tam)
```

Método que devuelve el contenido de la dirección de memoria pasada por parámetro

Parameters: `dir` - direccion a leer
`tam` - tamaño de la lectura: Byte, Word, Longword

Returns: Direccion de memoria leida

---

get

```
public java.lang.String get(long direccion)
```

Método que devuelve el contenido de la dirección de memoria pasada por parámetro con tamaño Long por defecto

Parameters: `direccion` - direccion a leer

Returns: Direccion de memoria leida

---

actualizaVIA

```
private void actualizaVIA(long direccion)
```

Método privado que actualiza la VIA según la dirección de memoria modificada

Parameters: `direccion` - dirección de memoria modificada, que puede corresponder con una dirección proyectada de la VIA

modificaMem

```
public void modificaMem(long direccion, java.lang.String valor,
                        char tam, boolean desdeTeclado)
```

Método que modifica la dirección de memoria pasada por parámetro

Parameters: `direccion` - direccion en la que escribir
`valor` - valor a escribir en la dirección
`tam` - tamaño de la escritura: Byte, Word, Longword

---

set

```
public void set(long direccion,java.lang.String valor,
                char tam,boolean desdeTeclado)
```

Método que escribe en la direccion de memoria pasada por parámetro, y actualiza la VIA

Parameters: `direccion` - direccion en la que escribir
`valor` - valor a escribir en la dirección
`tam` - tamaño de la escritura: Byte, Word, Longword

---

set

```
public void set(long direccion,java.lang.String valor,
                boolean desdeTeclado)
```

Método que escribe en la direccion de memoria pasada por parámetro con tamaño Long por defecto

Parameters: `direccion` - direccion en la que escribir
`valor` - valor a escribir en la dirección

---

getByteDireccion

```
private java.lang.String getByteDireccion(long b,long dir)
```

Método que devuelve el byte específico de la dirección pasada por parámetro llama a getValorByte para buscar dicho valor en el bloque adecuado

Parameters: `b` - direccion en el bloque de memoria
`dir` - direccion de memoria

Returns: byte con el contenido de la direccion de memoria indicada

---

getValorByte

```
private java.lang.String getValorByte(java.lang.String[] bloque,
                                      long b,long dir,extras.Par p)
```

Método que obtiene el valor del byte de memoria indicado

Parameters: `bloque` - bloque de memoria del que leemos
`b` - direccion en el bloque de memoria
`dir` - direccion de memoria
`p` -

Returns:

damePosicion

```
private extras.Par damePosicion(long direccion)
```

Método que a partir de una dirección decimal devuelve la posición y el bloque del bloque de memoria al que pertenece

Parameters: `direccion` - direccion de memoria
Returns: Pareja con el (num_bloque, dir_en_bloque)

---

dameBloque

```
private int dameBloque(long direccion)
```

Método que de una dir decimal devuelve el nº de bloque de memoria:
0-> Error
1-> Bloque1: excepciones
2-> Bloque2: RAM_monitor
3-> Bloque3: RAM_usuario
4-> Bloque4: noUsado1
5-> Bloque5: placaMultifuncionES
6-> Bloque6: noUsado2
7-> Bloque7: programaMonitor -> donde se aloja el MVE101Debug
8-> Bloque8: noUsado3

Parameters: `direccion` -
Returns: número del bloque

---

setByteDireccion

```
private void setByteDireccion(java.lang.String valor,long b,long dir,
                              boolean desdeTeclado)
```

Método que escribe el byte específico en la dirección pasada por parámetro
Llama a setValorByte para escribir dicho valor en el bloque adecuado

Parameters: `valor` - valor a escribir en la memoria
`b` - byte específico
`dir` - dirección pasada por parámetro
`desdeTeclado` - indica si ha sido reclamada desde teclado o no

---

protegidoMulti

```
boolean protegidoMulti(long dir)
```

Evalúa la dirección enviada por parámetro

Parameters: `dir` - dirección a examinar
Returns: Devuelve si la dirección es protegida o no

setValorByte

```
private void setValorByte(java.lang.String valor,
                          java.lang.String[] bloque,long b,long dir,
                          extras.Par p,boolean desdeTeclado)
```

Método que escribe el valor del byte en la dir y bloque indicados por parámetro

Parameters: `valor` - valor a escribir
`bloque` - bloque de memoria en el que escribir
`dir` - dirección del bloque en la que escribir
`b` - byte de la dirección en la que escribir
`p` - Par bloque-posición
`desdeTeclado` - reclamado desde teclado o no

---

setCircuito

```
public void setCircuito(perifericos.circuitoVIA.CircuitoVIA c)
```

Método para asociar la VIA proyectada en la memoria

Parameters: `c` - VIA proyectada en la memoria

---

getExcepciones

```
public java.lang.String[] getExcepciones()
```

Método para acceder al bloque de memoria de excepciones

Returns: bloque de memoria de excepciones

---

getRAM_monitor

```
public java.lang.String[] getRAM_monitor()
```

Método para acceder al bloque de memoria de RAM de monitor

Returns: bloque de memoria de RAM de monitor

---

getRAM_usuario

```
public java.lang.String[] getRAM_usuario()
```

Método para acceder al bloque de memoria de RAM de usuario

Returns: bloque de memoria de RAM de usuario

---

getPlacaMultifuncionES

```
public java.lang.String[] getPlacaMultifuncionES()
```

Método para acceder al bloque de memoria de la placa multifunción de E/S

Returns: bloque de memoria de la placa multifunción de E/S

Motorola →Memoria → hiloRetardo

motorola.memoria

Clase hiloRetardo

public class **hiloRetardo**
implements java.lang.Runnable

Clase que realiza la función del hilo de retardo del temporizador TL y TH
Title: **HILO RETARDO** Copyright (c) 2005

Detalle del constructor

hiloRetardo

```
public hiloRetardo(motorola.memoria.Memoria memoria,
                   int temporizador, int v)
```

Constructora de la clase **Hilo Retardo**

Parameters: `memoria` - Memoria
`temporizador` - tamaño del temporizador
`v` - tipo de procesador (lento, normal o rapido)

Detalles de los métodos

setVelocidad

```
public void setVelocidad(int v)
```

Método que modifica la velocidad del contador

Parameters: `v` - velocidad

setParteBaja

```
public void setParteBaja(long retardobajo)
```

Método para modificar la parte baja del temporizador TL

Parameters: `retardobajo` - retardo de la parte baja

---

setParteAlta

```
public void setParteAlta(long retardoalto)
```

Modifica la parte alta del temporizador TH y arranca el contador

Parameters: `retardoalto` - retardo de la parte alta

---

run

```
public void run()
```

Método run del hilo que activa los flag correspondiente en el temporizador, para implementar el retardo

Motorola → Registros

Este paquete implementara el Banco de Registros del **Motorola68k** con 8 registros de datos de 32 bits (accesibles en .B, .W o .L), 8 registros de direcciones de 32 bits (accesibles solo en .W o .L), 1 registro de estado de 16 bits y un contador de programa PC de 32 bits.

Motorola → Registros → BancoRegistros

motorola.registros

Clase BancoRegistros

public class **BancoRegistros**

Clase que representa el Banco de Registros con registros de datos D[8], registros de direcciones A[8], registro de estado SR[1] y contador de programa PC[1].
Title: **BANCO DE REGISTROS** Copyright (c) 2005

Detalle del constructor

BancoRegistros

`public` **`BancoRegistros`**`()`

Constructora de la clase **Bando de Registros**

Detalle del método

setRegDatos

`public void` **`setRegDatos`**`(int indice, long valor)`

Función que modifica un Registro de Datos dado, con un nuevo valor

Parameters: `indice` - Registro de Datos a modificar
`valor` - nuevo valor

SeleccionaRegDatos

`public motorola.registros.RegistroDatos` **`SeleccionaRegDatos`**`(int indice)`

Método que devuelve un Registro de Datos indicado

Parameters: `indice` - indice del Registro de Datos
Returns: Registro de Datos

MuestraRegDatos

`public java.lang.String` **`MuestraRegDatos`**`()`

Método que muestra el contenido de los Registro de Datos

Returns: muestra el contenido en hexadecimal de los registros de datos D

setRegDirecciones

`public void` **`setRegDirecciones`**`(int indice, long valor)`

Función que modifica un Registro de Direcciones dado, con un nuevo valor

Parameters: `indice` - Registro de Direcciones a modificar
`valor` - nuevo valor

SeleccionaRegDirecciones

`public motorola.registros.RegistroDirecciones`
`SeleccionaRegDirecciones``(int indice)`

Método que devuelve un Registro de Direcciones indicado

Parameters: `indice` - indice del Registro de Direcciones
Returns: Registro de Direcciones

MuestraRegDirecciones

`public java.lang.String` **`MuestraRegDirecciones`**`()`

Método que muestra el contenido de los Registro de Direcciones

Returns: muestra el contenido de los registros de direcciones en hexadecimal

---

getPC

`public motorola.registros.RegistroDirecciones` **`getPC`**`()`

Método que devuelve el Registro PC

Returns: contador de programa

FijaDireccionDeSaltoPC

`public void` **`FijaDireccionDeSaltoPC`**`(java.util.Vector Vsalto)`

Método que modifica el contenido del Registro PC con la dirección de salto

Parameters: `Vsalto` - direccion de salto del PC

---

setPC

`public void` **`setPC`**`(long valor)`

Función que modifica el contador de programa PC, con un nuevo valor

Parameters: `valor` - nuevo valor

---

getSR

`public motorola.registros.RegistroEstado` **`getSR`**`()`

Método que devuelve el Registro de Estado

Returns: registro de estado

getRegistrosDatos

```
public motorola.registros.RegistroDatos[] getRegistrosDatos()
```
Método accesor de los Registros de Datos
Returns: registros de datos

---

getRegistrosDirecciones

```
public motorola.registros.RegistroDirecciones[]
getRegistrosDirecciones()
```
Método accesor de los Registro de Direcciones
Returns: registros de direcciones

---

reseteaBancoRegistros

```
public void reseteaBancoRegistros()
```
Función que resetea el banco de registros haciendose uso de los reset de cada registro

---

reseteaRegistrosDatos

```
public void reseteaDatosRegistros()
```
Función que resetea los registros de datos D

---

reseteaRegistrosDirecciones

```
public void reseteaBancoRegistros()
```
Función que resetea los registros de direcciones

---

reseteaRegistroEstado

```
public void reseteaRegistroEstado()
```
Función que resetea el registro de estado SR

Motorola → Registros → RegistroDatos

motorola.registros

Clase RegistroDatos

public class **RegistroDatos**
extends motorola.registros.RegistroGeneral
Clase que implementa un registro de datos de 32 bits y su funcionamiento
Title: **REGISTRO DE DATOS** Copyright (c) 2005

Detalle del constructor

RegistroDatos

`public` **`RegistroDatos`**`()`

Constructora de la clase **Registro de datos**, su único parámetro es si es modificable

Detalles de los métodos

clearBit

`public void` **`clearBit`**`(int numBit)`

Método para limpiar un único bit

Parameters: `numBit` - número del bit a limpiar

---

clearRegistro

`public void` **`clearRegistro`**`(char tam)`

Método para limpiar un registro con el tamaño determinado por parámetro

Parameters: `tam` - tamaño de la operación

---

setBit

`public void` **`setBit`**`(int numBit, int valor)`

Método que modifica el valor de un bit concreto

Parameters: `numBit` - número del bit a modificar
`valor` - valor del bit

---

setRegistro

`public void` **`setRegistro`**`(char tam, java.util.Vector valor)`

Método que modifica el valor de un registro de datos

Parameters: `tam` - tamaño de la operación
`valor` - valor nuevo del registro

---

getbit

`public int` **`getbit`**`(int numBit)`

Método accesor para un bit determinado

Parameters: `numBit` - número de bit

Returns: bit

getRegistro

```
public java.util.Vector getRegistro(char tam)
```
Método accesor para un registro de datos
Parameters: `tam` - tamaño de la operación
Returns: registro de datos

Motorola → Registros → RegistroDirecciones

motorola.registros

Clase RegistroDirecciones

public class **RegistroDirecciones**
extends motorola.registros.RegistroGeneral
Implementa registros los de direcciones A de 32 bits accesibles en .W y .L
Title: **REGISTRO DE DIRECCIONES** Copyright (c) 2005

Detalle del constructor

RegistroDirecciones

```
public RegistroDirecciones()
```
Constructora de la clase **Registro de direcciones**

Detalles de los métodos

clearRegistro

```
public void clearRegistro(char tam)
```
Método para limpiar un registro con el tamaño determinado por parámetro
Parameters: `tam` - tamaño de la operación

---

setRegistro

```
public void setRegistro(char tam, java.util.Vector valor)
```
Método para modificar el valor de un registro de direcciones .W o .L
Parameters: `tam` - tamaño de la operación
`valor` - nuevo valor del registro de direcciones

---

setBit

```
public void setBit(int numBit, int valor)
```
Método para modificar el valor de un bit determinado
Parameters: `numBit` - número del bit a cambiar
`valor` - nuevo valor del bit

getRegistro

```
public java.util.Vector getRegistro(char tam)
```
Método para acceder al valor de un registro de direcciones .W o .L
Parameters: `tam` - tamaño de la operación
Returns: registro de direcciones

Motorola → Registros → RegistroEstado

motorola.registros

Clase RegistroEstado

public class **RegistroEstado**
Implementa un registro de estado de 16 bits y su funcionamiento. El usuario solo podrá acceder a los 8 últimos bits del registro y el administrador a todos
Title: **REGISTRO DE ESTADO** Copyright (c) 2005

Detalle del constructor

RegistroEstado

```
public RegistroEstado(boolean modo)
```
Constructora de la clase
Parameters: `modo` - modo usuario o administrador

Detalle del método

CambiaAUsuario

```
public void CambiaAUsuario()
```
Método para cambiar a modo usuario

CambiaAAdministrador

```
public void CambiaAAdministrador()
```
Método para cambiar a modo administrador

clear

```
public void clear()
```
Función que borra el contenido del registro de estado, poniendo su valor a '0'

setBit

`private void` **`setBit`**`(int numBit, int valor)`

Modifica el valor de un bit determinado. Para este registro solo nos importan las operaciones sobre los bits, ya que no habrá operaciones con otras longitudes

Parameters: `numBit` - número del bit
`valor` - nuevo valor

---

getBit

`private int` **`getBit`**`(int numBit)`

Método accesor a un bit

Parameters: `numBit` - número del bit
Returns: bit

---

setExtension

`public void` **`setExtension`**`(int valor)`

Método para modificar al bit de Extensión (el nº 4). Se denomina X

Parameters: `valor` - valor del bit

---

getExtension

`public int` **`getExtension`**`()`

Método accesor al bit de Extensión (el nº 4).

Returns: valor valor del bit

---

setNegativo

`public void` **`setNegativo`**`(int valor)`

Método para modificar al bit de Negativo (el nº 3). Se denomina N

Parameters: `valor` - valor del bit

---

getNegativo

`public int` **`getNegativo`**`()`

Método accesor al bit de Negativo (el nº 3).

Returns: valor del bit

---

setCero

`public void` **`setCero`**`(int valor)`

Método para modificar al bit de Cero (el nº 2). Se denomina Z

Parameters: `valor` - valor del bit

getCero

`public int` **`getCero`**`()`
Método accesor del bit de Cero (el nº 2).
Returns: valor valor del bit

---

setOverflow

`public void` **`setOverflow`**`(int valor)`
Método para modificar al bit de Overflow (el nº 1). Se denomina V
Parameters: `valor` - valor del bit

---

getOverflow

`public int` **`getOverflow`**`()`
Método accesor del bit de Overflow (el nº 1).
Parameters: `valor` - valor del bit

---

setCarry

`public void` **`setCarry`**`(int valor)`
Método para modificar al bit de Carry (el nº 0). Se denomina C
Parameters: `valor` - valor del bit

---

getCarry

`public int` **`getCarry`**`()`
Método accesor del bit de Carry (el nº 0).
Parameters: `valor` - valor del bit

---

setT1

`public void` **`setT1`**`(int valor)`
Función que actualiza el bit T1 (bit nº 15) correspondiente al modo de traza.
Solo accesible para modo administrador
Parameters: `valor` - nuevo valor

---

getT1

`public int` **`getT1`**`()`
Función que devuelve el bit T1
Returns: bit T1 (bit nº 15)

setT0

`public void **setT0**(int valor)`

Ael bit T0 (bit nº 14) correspondiente al modo de traza (modo administrador)

Parameters: `valor` - nuevo valor

---

getT0

`public int **getT0**()`

Función que devuelve el bit T0

Returns: bit T0 (bit nº 14)

---

setS

`public void **setS**(int valor)`

Actualiza el bit S (bit nº 13) correspondiente al estado del supervisor (modo administrador)

Parameters: `valor` - nuevo valor

---

getS

`public int **getS**()`

Función que devuelve el bit S

Returns: bit S (bit nº 13)

---

setM

`public void **setM**(int valor)`

Función que actualiza el bit M (bit nº 12) correspondiente al estado del supervisor. Solo accesible para modo administrador

Parameters: `valor` - nuevo valor

---

getM

`public int **getM**()`

Función que devuelve el bit N

Returns: bit M (bit nº 12)

---

setI2

`public void **setI2**(int valor)`

Función que actualiza el bit I2 (bit nº 10) correspondiente a la mascara de interrupciones. Solo accesible para modo administrador

Parameters: `valor` - nuevo valor

getI2

`public int` **`getI2`**`()`

Función que devuelve el bit I2

Returns: bit I2 (bit nº 10)

---

setI1

`public void` **`setI1`**`(int valor)`

Función que actualiza el bit I1 (bit nº 9) correspondiente a la mascara de interrupciones. Solo accesible para modo administrador

Parameters: `valor` - nuevo valor

---

getI1

`public int` **`getI1`**`()`

Función que devuelve el bit I1

Returns: bit I1 (bit nº 9)

---

setI0

`public void` **`setI0`**`(int valor)`

Función que actualiza el bit I0 (bit nº 8) correspondiente a la mascara de interrupciones. Solo accesible para modo administrador

Parameters: `valor` - nuevo valor

---

getI0

`public int` **`getI0`**`()`

Función que devuelve el bit I0

Returns: bit I0 (bit nº 8)

---

actualizaCondiciones

`public void` **`actualizaCondiciones`**`(java.util.Vector v)`

Método que actualiza el vector de condiciones del registro de estado depues de una operación de la ALU

Parameters: `v` - vector de condiciones nuevo

---

muestraContenido

`public java.lang.String` **`muestraContenido`**`()`

Método que muestra el contenido de un registro. Válida para cualquier tipo de registro

Returns:

Motorola → Registros → RegistroGeneral

motorola.registros

Clase RegistroGeneral

public abstract class **RegistroGeneral**

Clase que implementa un registro de general y su funcionamiento. Es abstracta, ya que son RegistroDatos y RegistroDirecciones quienes implementan todas sus funciones

Title: **REGISTRO GENERAL** Copyright (c) 2005

Detalle del constructor

RegistroGeneral

`public` **`RegistroGeneral`**`()`

Constructora de la clase **Registro general**

Detalle del método

muestraContenido

`public java.lang.String` **`muestraContenido`**`()`

Método que muestra el contenido de un registro. Válida para cualquier tipo de registro

Returns: contenido del registro en hexadecimal

3.3 Periféricos

Está formado por los subpaquetes **perifericos.circuitoVIA**, **perifericos.consola** y **perifericos.osciloscopio**, que representan al Circuito VIA, a la Consola de usuario y al Osciloscopio respectivamente.

Periféricos → CircuitoVIA →CircuitoVIA

perifericos.circuitoVIA

Clase CircuitoVIA

public class **CircuitoVIA**
extends perifericos.circuitoVIA.CircuitoVIAGUI

Clase que implementa al circuito VIA del MVME
Title: **CIRCUITO VIA** Copyright (c) 2005

Detalle del constructor

CircuitoVIA

`public CircuitoVIA(perifericos.consola.Consola c)`

Constructora de la clase **CircuitoVIA**

Parameters: `c` - Consola asociada con la que se inicializa la VIA.

Detalles de los métodos

ActualizaViayMuestra

`public void ActualizaViayMuestra()`

Función que actualiza el contenido de los registros de control de la VIA y sus correspondientes direcciones de memoria. Al finalizar la actualización se dibujan todos los elementos de la VIA.

Returns:

---

dibujaElementos

`public void dibujaElementos(int actualizaLEDsDisplaysTodo)`

Función que dibuja los led's de la VIA1: mediante el puerto ORB accedemos al teclado matricial, y con el puerto ORA a los led's(4-7) y los switches(0-3). Y los 8 displays de 7-segmentos de la VIA2: mediante el puerto ORA indicamos los display's que se encenderán (solo uno de cada vez), y con el puerto ORB los segmentos a mostrar del display.

boton0_mousePressed

```
public void boton0_mousePressed(MouseEvent e)
```

Metodo para capturar la pulsación del botón 0

Parameters: e -

---

boton0_mouseReleased

```
public void boton0_mouseReleased(MouseEvent e)
```

Metodo para capturar la depresión del botón 0

Parameters: e -

---

boton1_mousePressed

```
public void boton1_mousePressed(MouseEvent e)
```

Metodo para capturar la pulsación del botón 1

Parameters: e -

---

boton1_mouseReleased

```
public void boton1_mouseReleased(MouseEvent e)
```

Metodo para capturar la depresión del botón 1

Parameters: e -

---

boton2_mousePressed

```
public void boton2_mousePressed(MouseEvent e)
```

Metodo para capturar la pulsación del botón 2

Parameters: e -

---

boton2_mouseReleased

```
public void boton2_mouseReleased(MouseEvent e)
```

Metodo para capturar la depresión del botón 2

Parameters: e -

---

boton3_mousePressed

```
public void boton3_mousePressed(MouseEvent e)
```

Metodo para capturar la pulsación del botón 3

Parameters: e –

boton3_mouseReleased

```
public void boton3_mouseReleased(MouseEvent e)
```

Metodo para capturar la depresión del botón 3

Parameters: e -

---

boton4_mousePressed

```
public void boton4_mousePressed(MouseEvent e)
```

Metodo para capturar la pulsación del botón 4

Parameters: e -

---

boton4_mouseReleased

```
public void boton4_mouseReleased(MouseEvent e)
```

Metodo para capturar la depresión del botón 4

Parameters: e -

---

boton5_mousePressed

```
public void boton5_mousePressed(MouseEvent e)
```

Metodo para capturar la pulsación del botón 5

Parameters: e -

---

boton5_mouseReleased

```
public void boton5_mouseReleased(MouseEvent e)
```

Metodo para capturar la depresión del botón 5

Parameters: e -

---

boton6_mousePressed

```
public void boton6_mousePressed(MouseEvent e)
```

Metodo para capturar la pulsación del botón 6

Parameters: e -

---

boton6_mouseReleased

```
public void boton6_mouseReleased(MouseEvent e)
```

Metodo para capturar la depresión del botón 6

Parameters: e –

boton7_mousePressed

`public void` **`boton7_mousePressed`**`(MouseEvent e)`

Metodo para capturar la pulsación del botón 7

Parameters: `e` -

---

boton7_mouseReleased

`public void` **`boton7_mouseReleased`**`(MouseEvent e)`

Metodo para capturar la depresión del botón 7

Parameters: `e` -

---

boton8_mousePressed

`public void` **`boton8_mousePressed`**`(MouseEvent e)`

Metodo para capturar la pulsación del botón 8

Parameters: `e` -

---

boton8_mouseReleased

`public void` **`boton8_mouseReleased`**`(MouseEvent e)`

Metodo para capturar la depresión del botón 8

Parameters: `e` -

---

boton9_mousePressed

`public void` **`boton9_mousePressed`**`(MouseEvent e)`

Metodo para capturar la pulsación del botón 9

Parameters: `e` -

---

boton9_mouseReleased

`public void` **`boton9_mouseReleased`**`(MouseEvent e)`

Metodo para capturar la depresión del botón 9

Parameters: `e` -

---

botonA_mousePressed

`public void` **`botonA_mousePressed`**`(MouseEvent e)`

Metodo para capturar la pulsación del botón A

Parameters: `e` -

botonA_mouseReleased

```
public void botonA_mouseReleased(MouseEvent e)
```

Metodo para capturar la depresión del botón A

Parameters: e -

---

botonB_mousePressed

```
public void botonB_mousePressed(MouseEvent e)
```

Metodo para capturar la pulsación del botón B

Parameters: e -

---

botonC_mouseReleased

```
public void botonC_mouseReleased(MouseEvent e)
```

Metodo para capturar la depresión del botón C

Parameters: e -

---

botonC_mousePressed

```
public void botonC_mousePressed(MouseEvent e)
```

Metodo para capturar la pulsación del botón C

Parameters: e -

---

botonD_mouseReleased

```
public void botonD_mouseReleased(MouseEvent e)
```

Metodo para capturar la depresión del botón D

Parameters: e -

---

botonAlmohadilla_mousePressed

```
public void botonAlmohadilla_mousePressed(MouseEvent e)
```

Metodo para capturar la pulsación del botón #

Parameters: e -

---

botonAlmohadilla_mouseReleased

```
public void botonAlmohadilla_mouseReleased(MouseEvent e)
```

Metodo para capturar la depresión del botón #

Parameters: e –

botonAsterisco_mousePressed

`public void` **`botonAsterisco_mousePressed`**`(MouseEvent e)`

Metodo para capturar la pulsación del botón *

Parameters: `e` -

---

botonAsterisco_mouseReleased

`public void` **`botonAsterisco_mouseReleased`**`(MouseEvent e)`

Metodo para capturar la depresión del botón *

Parameters: `e` -

---

getVIA1

`public perifericos.circuitoVIA.CircuitoVIA1` **`getVIA1`**`()`

Método que devuelve la VIA1

Returns: VIA1

---

getVIA2

`public perifericos.circuitoVIA.CircuitoVIA2` **`getVIA2`**`()`

Método que devuelve la VIA2

Returns: VIA2

---

switch1_actionPerformed

`public void` **`switch1_actionPerformed`**`(java.awt.event.ActionEvent e)`

Metodo para que cuando se modifique el Switch1 se refleje su estado en el registro ORA[0]

Parameters: `e` -

---

switch2_actionPerformed

`public void` **`switch2_actionPerformed`**`(java.awt.event.ActionEvent e)`

Metodo para que cuando se modifique el Switch2 se refleje su estado en el registro ORA[1]

Parameters: `e` -

---

switch3_actionPerformed

`public void` **`switch3_actionPerformed`**`(java.awt.event.ActionEvent e)`

Metodo para que cuando se modifique el Switch3 se refleje su estado en el registro ORA[2]

Parameters: `e` -

switch4_actionPerformed

`public void` **`switch4_actionPerformed`**`(java.awt.event.ActionEvent e)`

Metodo para que cuando se modifique el Switch4 se refleje su estado en el registro ORA[3]

Parameters: `e` –

---

nand

`private boolean` **`nand`**`()`

Método auxiliar que implementa la función NAND de 4 elementos

Returns:

---

detectaTransicion

`private void` **`detectaTransicion`**`(boolean transAnterior)`

Método auxiliar para detectar el tipo de transición y si ha ocurrido o no

Parameters: `transAnterior` - estado anterior de CB1

---

botonMousePressed

`private void` **`botonMousePressed`**`(int fila,int columna)`

Metodo para que cuando pulsemos un boton se refleje su estado en el registro ORB[fila] y ORB[columna]

Parameters: `fila` - numero de fila
`columna` - numero de columna

---

botonMouseReleased

`private void` **`botonMouseReleased`**`(int fila,int columna)`

Metodo para que cuando soltemos un boton se refleje su estado en el registro ORB[fila] y ORB[columna]

Parameters: `fila` - numero de fila
`columna` - numero de columna

---

botonAbort_actionPerformed

`public void` **`botonAbort_actionPerformed`**`(java.awt.event.ActionEvent e)`

Método del botón Abort de la VIA. Se muestra el contenido de los registros, se resetean las posiciones de memoria y los breakpoints y se detiene el hilo de ejecución.

Parameters: `e` -

botonReset_actionPerformed

```
public void botonReset_actionPerformed(java.awt.event.ActionEvent e)
```

Método para resetear el contenido de la memoria. Se resetean las direcciones de memoria de la placa multifunción y los registros, además de parar el hilo.

Parameters: e –

Perifericos → CircuitoVIA → CircuitoVIA1

perifericos.circuitoVIA

Clase CircuitoVIA1

public class **CircuitoVIA1**
implements perifericos.circuitoVIA.CircuitoVIAGenerico

Clase que implementa el circuito VIA 1 (teclado matricial, led's y switches)
Title: **CIRCUITO VIA 1** Copyright (c) 2005

Detalle del constructor

CircuitoVIA1

```
public CircuitoVIA1()
```

Constructor de la VIA1, ORB ==>teclado matricial 4x4;
ORA ==> [0-3]:switches ; [4-7]:led's

Detalles de los métodos

modificaPuerto

```
public void modificaPuerto(perifericos.circuitoVIA.Puerto p,
                           int linea,int valor)
```

Función que modifica el valor de una linea de un puerto indicado

Parameters: p - puerto a modificar ORA u ORB
linea - linea del puerto a modificar
valor - nuevo valor de la linea

---

modificaEntradaSalida

```
public void modificaEntradaSalida(perifericos.circuitoVIA.Puerto p,
                                  int linea, int es)
```

Función que modifica la capacitación de E/S de una linea de un puerto

Parameters: p - puerto a modificar ORA u ORB
linea - linea del puerto a modificar
es - nuevo valor de la capacitación de E/S

reseteaPuerto

`public void` **`reseteaPuerto`**`(perifericos.circuitoVIA.Puerto p)`

Función que resetea el contenido de un puerto poniendolo a ‘0’

Parameters: `p` -

---

actualizaORB

`public void` **`actualizaORB`**`(java.lang.String contenido)`

Función que actualiza el valor del puerto ORB

Parameters: `contenido` - nuevo contenido

---

actualizaORA

`public void` **`actualizaORA`**`(java.lang.String contenido)`

Función que actualiza el valor del puerto ORA

Parameters: `contenido` - nuevo contenido

---

actualizaDDRB

`public void` **`actualizaDDRB`**`(java.lang.String contenido)`

Función que actualiza el valor del puerto DDRB

Parameters: `contenido` - nuevo contenido

---

actualizaDDRA

`public void` **`actualizaDDRA`**`(java.lang.String contenido)`

Función que actualiza el valor del puerto DDRA

Parameters: `contenido` - nuevo contenido

---

actualizaPCR

`public void` **`actualizaPCR`**`(java.lang.String contenido)`

Método que detecta las transiciones de los puertos de la VIA

Parameters: `contenido` – nuevo contenido

---

getPuertoORA

`public perifericos.circuitoVIA.Puerto` **`getPuertoORA`**`()`

Método que devuelve el puerto ORA

Returns: puerto ORA

getPuertoORB

```
public perifericos.circuitoVIA.Puerto getPuertoORB()
```

Método que devuelve el puerto ORB

Returns: puerto ORB

Perifericos → CircuitoVIA → CircuitoVIA2

perifericos.circuitoVIA

Clase CircuitoVIA2

public class **CircuitoVIA2**
implements perifericos.circuitoVIA.CircuitoVIAGenerico

Clase que implementa el circuito VIA 2 (displays 7 segmentos)
Title: **CIRCUITO VIA 2** Copyright (c) 2005

Detalle del constructor

CircuitoVIA2

```
public CircuitoVIA2(int x, int y)
```

Constructora de la VIA2: ORB==> display encendido;
ORA==> codigo 7-segmentos del display

Parameters: `x` - coordenada X
`y` - coordenada Y

Detalles de los métodos

modificaPuerto

```
public void modificaPuerto(perifericos.circuitoVIA.Puerto p,
                           int linea, int valor)
```

Función que modifica el valor de una linea de un puerto indicado

Parameters: `p` - puerto a modificar ORA u ORB
`linea` - linea del puerto a modificar
`valor` - nuevo valor de la linea

---

modificaEntradaSalida

```
public void modificaEntradaSalida(perifericos.circuitoVIA.Puerto p,
                                  int linea, int es)
```

Función que modifica la capacitación de E/S de una linea de un puerto

Parameters: `p` - puerto a modificar ORA u ORB
`linea` - linea del puerto a modificar
`es` - nuevo valor de la capacitación de E/S

reseteaPuerto

`public void` **`reseteaPuerto`**`(perifericos.circuitoVIA.Puerto p)`

Función que resetea el contenido de un puerto

Parameters: `p` -

---

actualizaORB

`public void` **`actualizaORB`**`(java.lang.String contenido)`

Función que actualiza el valor del puerto ORB

Parameters: `contenido` - nuevo contenido

---

actualizaORA

`public void` **`actualizaORA`**`(java.lang.String contenido)`

Función que actualiza el valor del puerto ORA

Parameters: `contenido` - nuevo contenido

---

actualizaDDRB

`public void` **`actualizaDDRB`**`(java.lang.String contenido)`

Función que actualiza el valor del puerto DDRB

Parameters: `contenido` - nuevo contenido

---

actualizaDDRA

`public void` **`actualizaDDRA`**`(java.lang.String contenido)`

Función que actualiza el valor del puerto DDRA

Parameters: `contenido` - nuevo contenido

---

getPuertoORA

`public perifericos.circuitoVIA.Puerto` **`getPuertoORA`**`()`

Método accesor al puerto ORA

Returns: puerto ORA

---

getPuertoORB

`public perifericos.circuitoVIA.Puerto` **`getPuertoORB`**`()`

Método accesor al puerto ORB

Returns: puerto ORB

getDisplay0

```
public perifericos.circuitoVIA.DisplaySegmentos getDisplay0()
```
Método accesor al display 0
Returns: display 0

---

getDisplay1

```
public perifericos.circuitoVIA.DisplaySegmentos getDisplay1()
```
Método accesor al display 1
Returns: display 1

---

getDisplay2

```
public perifericos.circuitoVIA.DisplaySegmentos getDisplay2()
```
Método accesor al display 2
Returns: display 2

---

getDisplay3

```
public perifericos.circuitoVIA.DisplaySegmentos getDisplay3()
```
Método accesor al display 3
Returns: display 3

---

getDisplay4

```
public perifericos.circuitoVIA.DisplaySegmentos getDisplay4()
```
Método accesor al display 4
Returns: display 4

---

getDisplay5

```
public perifericos.circuitoVIA.DisplaySegmentos getDisplay5()
```
Método accesor al display 5
Returns: display 5

---

getDisplay6

```
public perifericos.circuitoVIA.DisplaySegmentos getDisplay6()
```
Método accesor al display 6
Returns: display 6

getDisplay7

```
public perifericos.circuitoVIA.DisplaySegmentos getDisplay7()
```
Método accesor al display 7
Returns: display 7

Periféricos → CircuitoVIA → CircuitoVIAGenerico

perifericos.circuitoVIA

Interfaz CircuitoVIAGenerico

public interface **CircuitoVIAGenerico**
Clase que representa un circuito VIA genérico
Title: **CIRCUITO VIA GENERICO** Copyright (c) 2005

Detalles de los métodos

modificaPuerto

```
public void modificaPuerto(perifericos.circuitoVIA.Puerto p,
                           int linea,int valor)
```
Función que modifica el valor de una linea de un puerto indicado
Parameters: `p` - puerto a modificar ORA u ORB
`linea` - linea del puerto a modificar
`valor` - nuevo valor de la linea

---

modificaEntradaSalida

```
public void modificaEntradaSalida(perifericos.circuitoVIA.Puerto p,
                                  int linea,int es)
```
Función que modifica la capacitación de E/S de una linea de un puerto
Parameters: `p` - puerto a modificar ORA u ORB
`linea` - linea del puerto a modificar
`es` - nuevo valor de la capacitación de E/S

---

reseteaPuerto

```
public void reseteaPuerto(perifericos.circuitoVIA.Puerto p)
```
Función que resetea el contenido de un puerto
Parameters: `p` - puerto a resetear

---

actualizaORB

```
public void actualizaORB(java.lang.String contenido)
```
Función que actualiza el valor del puerto ORB
Parameters: `contenido` - nuevo contenido

actualizaORA

`public void` **`actualizaORA`**`(java.lang.String contenido)`

Función que actualiza el valor del puerto ORA

Parameters: `contenido` - nuevo contenido

---

actualizaDDRB

`public void` **`actualizaDDRB`**`(java.lang.String contenido)`

Función que actualiza el valor del puerto DDRB

Parameters: `contenido` - nuevo contenido

---

actualizaDDRA

`public void` **`actualizaDDRA`**`(java.lang.String contenido)`

Función que actualiza el valor del puerto DDRA

Parameters: `contenido` - nuevo contenido

---

getPuertoORA

`public perifericos.circuitoVIA.Puerto` **`getPuertoORA`**`()`

Método que devuelve el puerto ORA

Returns: puerto ORA

---

getPuertoORB

`public perifericos.circuitoVIA.Puerto` **`getPuertoORB`**`()`

Método que devuelve el puerto ORB

Returns: puerto ORB

Perifericos → CircuitoVIA → CircuitoVIAGUI

perifericos.circuitoVIA

Clase CircuitoVIAGUI

public abstract class **CircuitoVIAGUI**
extends javax.swing.JInternalFrame

Interfaz gráfico del circuito VIA

Title: **CIRCUITO VIA GUI** Copyright (c) 2005

Detalle del constructor

CircuitoVIAGUI

```
public CircuitoVIAGUI()
```

Constructora del interfaz gráfico de la clase CircuitoVIA. Todos sus métodos son abstractos, los implementa la clase CircuitoVIA, solamente tiene una función para inicializar todos sus componentes.

Detalles de los métodos

jbInit

```
private void jbInit()
```

Método que inicia los componentes gráficos del interfaz

Lanza: `Exception` - error en la inicialización de los componentes gráficos

Perifericos → CircuitoVIA → DisplaySegmentos

perifericos.circuitoVIA

Clase DisplaySegmentos

class **DisplaySegmentos**

Clase que representa a los displays de 7 segmentos
Title: **DISPLAY SEGMENTOS** Copyright (c) 2005

Detalle del constructor

DisplaySegmentos

```
public DisplaySegmentos(int x, int y)
```

Constructora de la clase **DisplaySegmentos**. A partir de las coordenadas iniciales, creamos los segmentos con sus coordenadas genéricas.

Parameters: `x` - coordenada X del display
`y` - coordenada Y del display

Detalles de los métodos

dibuja

```
public void dibuja(perifericos.circuitoVIA.CircuitoVIAGUI VIA,
                   java.util.Vector segmentosDib)
```

Función que dibuja un display a partir del vector de segmentos (puerto ORA)

Parameters: `VIA` -
`segmento` - vector de segmentos (puerto ORA de la VIA) con los segmentos a dibujar

getSegmentoA

```
public extras.Par getSegmentoA()
```
segmento A
Returns: segmento A

---

getSegmentoB

```
public extras.Par getSegmentoB()
```
segmento B
Returns: segmento B

---

getSegmentoC

```
public extras.Par getSegmentoC()
```
segmento C
Returns: segmento C

---

getSegmentoD

```
public extras.Par getSegmentoD()
```
segmento D
Returns: segmento D

---

getSegmentoE

```
public extras.Par getSegmentoE()
```
segmento E
Returns: segmento E

---

getSegmentoF

```
public extras.Par getSegmentoF()
```
segmento F
Returns: segmento F

---

getSegmentoG

```
public extras.Par getSegmentoG()
```
segmento G
Returns: segmento G

getPunto

`public java.awt.Point` **`getPunto`**`()`
punto del display
Returns: punto

Periféricos → CircuitoVIA → Puerto

perifericos.circuitoVIA

Clase Puerto

public class **Puerto**
Clase que implementa los puertos ORA y ORB con sus respectivos DDRA y DDRB de la MVME
Title: **PUERTO** Copyright (c) 2005

Detalle del constructor

Puerto

`public` **`Puerto`**`()`
Constructora de la clase **Puerto** que inicializa sus variables

Detalles de los métodos

habilitarLineaEscrituraLectura

`public void` **`habilitarLineaEscrituraLectura`**`(int linea, int EoL)`
Función que habilita la E/S de un puerto, modificando el valor del registro DDR
Parameters: `linea` - linea del registro que se modifica
`EoL` - valor de E/S del registro

---

modificarLineaEntrada

`public void` **`modificarLineaEntrada`**`(int linea,int valor)`
Funcion que modifica la linea del puerto si está habilitada para escritura
Parameters: `linea` - linea del puerto a modificar
`valor` - nuevo valor de la linea

---

obtenerLineaSalida

`public int` **`obtenerLineaSalida`**`(int linea)`
Función que obtiene el valor de una linea del puerto indicado
Parameters: `linea` - linea del puerto a consultar
Returns: valor de la línea solicitada

setDireccionMemoria

`public void` **`setDireccionMemoria`**`(java.lang.String direccion)`

Función que modifica la direccion de memoria de un puerto

Parameters: `direccion` – dirección de memoria del puerto

---

getRegDDR

`public java.util.Vector` **`getRegDDR`**`()`

Función que devuelve el registro DDR de control del puerto

Returns: registro DDR

---

getPuertoP

`public java.util.Vector` **`getPuertoP`**`()`

Función que devuelve el puerto

Returns: Vector del puerto

---

getDireccionMemoria

`public java.lang.String` **`getDireccionMemoria`**`()`

Función que devuelve la dirección de memoria del puerto

Returns: dirección de memoria del puerto

Perifericos → Consola → Consola

perifericos.circuitoVIA

Clase Consola

public class **Consola** extends ConsolaGUI

Clase que implementa las funciones de la consola del MVME

Title:**CONSOLA**

Detalles del constructor

Consola

`public` **`Consola`**`()`

Constructora de la clase **Consola**

Detalles de los métodos

TrazarUna

`private void` **`TrazarUna`**`(String cadenaEdicion, boolean primeraTraza)`

Método que realiza una traza del programa

Parameters: `cadenaEdicion` - operandos para la traza
`primeraTraza` - indica si es la primera traza o no

---

editarMemoria

`private void` **`editarMemoria`**`(String nuevoValor)`

Método para editar la memoria desde la consola

Parameters: `nuevoValor` - valor de memoria nuevo

---

jTextFieldConsola_keyPressed

`public void` **`jTextFieldConsola_keyPressed`**`(KeyEvent e)`

Función que reconoce la acción de pulsación de una tecla

Parameters: `e` - evento de teclado

---

setModoEditar

`public void` **`setModoEditar`**`(boolean edicion,long dir, boolean des)`

Método que activa/desactiva el modo de edición de memoria a partir de una direccion

Parameters: `edicion` - activación/desactivación del modo edición
`dir` - direcciόm de comienzo de la edición

---

getMVME101

`public MVME101` **`getMVME101`**`()`

Método accesor para la MVME101

Returns: Devuelve el MVME101 asociado al terminal de consola

---

setCircuito

`public void` **`setCircuito`**`(CircuitoVIA c)`

Método que asocia la VIA a la Memoria

Parameters: `c` - CircuitoVIA a asociar a la memoria.

Perifericos → Consola →ConsolaGUI

perifericos.circuitoVIA

Clase ConsolaGUI

public abstract class **ConsolaGUI**
extends JInternalFrame

Interfaz gráfico de la consola.
Title:**CONSOLA GUI**

Detalles del constructor

ConsolaGUI

`public ConsolaGUI()`

Constructora del interfaz gráfico de la clase Consola

Detalles de los métodos

jbInit

`private void jbInit()`

Método que inicia los componentes gráficos del interfaz

---

getTextFieldConsola

`public JTextComponent getTextFieldConsola()`

Método que devuelve el contenido del que introducimos en la consola
Returns: Texto por consola

---

escribeEnConsola

`public void escribeEnConsola(String cadena)`

Método que escribe por consola
Parameters: `cadena` - Texto a escribir por consola

Perifericos → Consola → S_Record

perifericos.consola

Clase S_Record

public class **S_Record**

Clase para representar el código S-Record codificado por Motorola para la creación del del archivo ".HEX" que interpretará el programa
Title:**SRECORD**

Detalles del constructor

S_Record

`public` **`S_Record`**`(String srecord)`

Constructora de la clase **S-Record** a partir de un string con la cadena a formar

Parameters: `srecord` - cadena con la que formar el S-Record

Detalles de los métodos

getType

`public String` **`getType`**`()`

Método accesor para el tipo

Returns: tipo del s_record

---

getLength

`public String` **`getLength`**`()`

Método accesor para la longitud

Returns: longitud del s_record

---

getAddress

`public String` **`getAddress`**`()`

Método accesor para la dirección

Returns: dirección del srecord

---

getCodeData

`public String` **`getCodeData`**`()`

Método accesor para el código de datos

Returns: sección de datos del srecord

---

getChecksum

`public String` **`getChecksum`**`()`

Método accesor para el checksum

Returns: checksum del srecord

getToCompareWithChecksum

`public String` **`getToCompareWithChecksum`**`()`
Método que calcula el checksum del s-record
Returns: checksum del s-record

Perifericos → Consola → MVME101

perifericos.consola

Clase MVME101

public class **MVME101**
Clase para representar al periférico MVME101
Title:**MVME101**

Detalles del Constructor

MVME101

`public` **`MVME101`**`()`
Constructora de la clase **MVME101**

Detalles de los métodos

analiza

`public String` **`analiza`**`(String instruccion, ConsolaGUI consola)`
Función que segun la instrucción pasada por parámetro realiza una u otra tarea
Parameters: `instruccion` - instruccion por parámetro
Returns: cadena con el resultado de la acción de dicha instrucción

---

leerOpciones

`private void` **`leerOpciones`**`(Set opciones, String textoOpciones)`
Devuelve un conjunto con todas las cadenas separadas por espacios susceptibles de ser opciones
Parameters: `opciones` - conjunto de cadenas suceptibles de ser opciones
`textoOpciones` - cadena de texto de la cual se extraen las posibles opciones

---

compruebaChecksum

`private boolean` **`compruebaChecksum`**`(S_Record srecord)`
Función que comprueba el checksum en un String srecord
Parameters: `srecord` en el cual se va a realizar la comprobación de checksum
Returns: booleano que indica si es correcto el cheksum o no

loadMemory

```
private String loadMemory(String operandos)
```

The LO command formats s-records received at Serial Port2 in object data and stores it in the memory locations specified in the s-records. ignores checksum while loading X echoes received data to console : is an optional string to be transmitted through Serial Port 2 before loading

---

cargarPrograma

```
private String cargarPrograma(boolean checksum, boolean echoes,
                              String texto)
```

Función que carga el programa en memoria

Parameters: `checksum` - indica si se ha activado la opción de comprobar checksum
`echoes` - indica si se ha activado la opción de mostrar eco por consola
`texto` - programa a cargar en memoria

Returns:

---

carga

```
private String carga(S_Record str)
```

Función que carga el S-Record indicado en memoria en formato binario

Parameters: `str` - srecord a cargar en memoria
Returns: posible error

---

displaySetAddress

```
private String displaySetAddress(String operandos,boolean estiloTraza)
```

Función que muestra o modifica el contenido de los registros de direcciones

Parameters: `operandos` - nuevo contenido del registro a modificar
Returns: posible error

---

displaySetData

```
private String displaySetData(String operandos, boolean estiloTraza)
```

Función que muestra o modifica el contenido de los registros de datos

Parameters: `operandos` - nuevo contenido del registro a modificar
Returns: posible error

---

displaySetPC

```
private String displaySetPC(String operandos, boolean estiloTraza)
```

Función que muestra o modifica el contenido del contador de programa

Parameters: `operandos` - nuevo contenido del contador de programa
Returns: posible error

displaySetStatus

```
private String displaySetStatus(String operandos,boolean estiloTraza)
```

Función que muestra o modifica el contenido del registro de estado

Parameters: `operandos` - nuevo contenido del registro de estado
Returns: posible error

---

displaySetUserStack

```
private String displaySetUserStack(String operandos,
                                   boolean estiloTraza)
```

Función que muestra o modifica el contenido de la pila de usuario

Parameters: `operandos` - nuevo contenido de la pila de usuario
Returns: posible error

---

displaySetSupervisorStack

```
private String displaySetSupervisorStack(String operandos,
                                         boolean estiloTraza)
```

Función que muestra o modifica el contenido de la pila de supervisor

Parameters: `operandos` - nuevo contenido de la pila de supervisor
Returns: posible error

---

displaySetGenerico

```
private String displaySetGenerico(char registro,String operandos,
                                  boolean estiloTraza)
```

Función que muestra o modifica el contenido del registro indicado

Parameters: `registro` - registro indicado (dirección, datos, PC, SR, US, SS)
`operandos` - nuevo contenido
Returns: posible error

---

mostrarRegistros

```
public String mostrarRegistros(char tipoReg, boolean todos,
                               int num, boolean estiloTraza)
```

Método que muestra por consola el contenido de los registros de dirección o datos

Parameters: `tipoReg` - tipo del registro a mostrar (direcciones, datos, estado, ...)
`num` - numero de registros a mostrar en el caso de reg de direcciones y datos (todos sino se muestra solo el indicado)
Returns: posible error

memoryDisplay

```
private String memoryDisplay(String operandos)
```

Función que lee el contenido de la memoria y lo interpreta indicandolo al usuario

Parameters: `operandos` - operandos de la instrucción
Returns: posible error

---

desensamblaDireccion

```
public Par desensamblaDireccion(long direccion)
```

Método que desensambla una determinada dirección de la memoria

Parameters: `direccion` - dirección a desensamblar
Returns: Par con el contenido desensamblado y el valor de la dirección

---

leerPrograma

```
private String leerPrograma(boolean desensamblado,
                            long direccionComienzo, int numeroBytes)
```

Función que lee el contenido de la memoria

Parameters: `desensamblado` - opciones de desensamblado de la memoria
`direccionComienzo` - dirección de comienzo de la lectura de memoria
`numeroBytes` - numero de bytes a leer de memoria
Returns: posible error

---

memoryModify

```
private String memoryModify(String operandos)
```

Función que modifica el contenido de la memoria y lo interpreta indicandolo al usuario

Parameters: `operandos` - operandos de la instrucción
Returns: posible error

---

modificarMemoria

```
private String modificarMemoria(boolean desensamblado,
                                long direccionComienzo,char sizeData,
                                boolean notReadData)
```

Función que modifica la memoria según unos opciones determinadas

Parameters: `desensamblado` - opciones de desensamblado
`direccionComienzo` - direccion de comienzo
`sizeData` - tamaño a leer
`notReadData` - solo escritura
Returns:posible error

goExecute

`private String` **`goExecute`**`(String operandos)`

Función que crea el hilo de ejecución del programa ensamblador

Parameters:`operandos` - dirección de comienzo de la ejecución del programa
Returns:posible error

---

goExecuteBreakpoint

`private String` **`goExecuteBreakpoint`**`(String operandos)`

Método para realizar la ejecución del programa hasta una dirección dada

Parameters:`operandos` - dirección de finalización de la ejecución
Returns:posible error

---

ejecuta

`private String` **`ejecuta`**`(String operandos, boolean gt)`

Método que ejecuta el programa para GO o GT

Parameters: `operandos` - operandos de la ejecución
`gt` - indica si la ejecución tiene una dirección de finalización
Returns:posible error

---

traceOne

`public String` **`traceOne`**`(String operandos, boolean esPrimera)`

Función que realiza la ejecución de una sola instrucción del programa

Parameters: `operandos` - número de trazas a ejecutar
`esPrimera` - una única traza
Returns:posible error

---

breakpointSet

`private String` **`breakpointSet`**`(String operandos)`

Método que añade breakpoints para la ejecución

Parameters:`operandos` - operandos de la instrucción
Returns:posible error

---

mostrarTablaBreakPoints

`private void` **`mostrarTablaBreakPoints`**`()`

Método para mostrar la tabla de breakpoints por consola

ordenaDirecciones

`private void` **`ordenaDirecciones`**`(Object[] direcciones)`

Método para ordenar la tabla de breakpoints usando el algoritmo de selección de minimo

Parameters:`direcciones` - direcciones de breakpoints

---

calculaMinimo

`private int` **`calculaMinimo`**`(Object[] direcciones,int i, int j)`

Método para calcular el minimo

Parameters: `direcciones` - tabla de breakpoints
`i` - inicial
`j` - final

Returns: minimo

---

breakpointRemove

`private String` **`breakpointRemove`**`(String operandos)`

Método que elimina los breakpoints añadidos anteriormente

Parameters:`operandos` - distintos breakpoints
Returns:posible error

---

help

`private String` **`help`**`(String operandos)`

Función que muestra el menu de ayuda según unos opciones determinadas

Parameters:`operandos` - opciones extra
Returns:posible error

Perifericos → Osciloscopio

Paquete Osciloscopio

El paquete referente al osciloscopio esta compuesto por todas las clases mediante las cuales se gestiona y se muestra el comportamiento de la simulación del osciloscopio.

Perifericos → Osciloscopio → Osciloscopio

perifericos.osciloscopio

Clase Osciloscopio

public class **Osciloscopio**
extends perifericos.osciloscopio.OsciloscopioGUI

Clase que implementa el osciloscopio
Title:**OSCILOSCOPIO** Copyright (c) 2005

Detalle del constructor

Osciloscopio

`public` **`Osciloscopio`**`()`

Constructora de la clase **Osciloscopio**

Detalles de los métodos

getPrimeraVez

`public boolean` **`getPrimeraVez`**`()`

Returns:

---

setPunto

`public void` **`setPunto`**`(int x,int y)`

Método que activa (enciende) una coordenada de la matriz de puntos [256,256] de la pantalla del osciloscopio (origen coord: abajo-izquierda)

Parameters: `x` - coordenada X
`y` - coordenada Y

---

paint

`public void` **`paint`**`()`

Redefinimos el método paint del osciloscopio

---

refrescaPantalla

`public void` **`refrescaPantalla`**`(boolean limpia)`

Función que refresca la pantalla del osciloscopio

Parameters: `limpia` - indica si se tiene que limpiar la pantalla del osciloscopio

---

muestraPantallaInicial

`public void` **`muestraPantallaInicial`**`()`

Método que muestra la pantalla de inicio de encendido del osciloscopio

damePantallainicial

`public java.awt.Graphics` **`damePantallainicial`**`()`
Método accesor a la pantalla inicial
Returns: Pantalla

---

modificaIluminacion

`private void` **`modificaIluminacion`**`(float ilum)`
Método que modifica la ilumincación de la pantalla del osciloscopio
Parameters: `ilum` – índice de iluminacion

---

botonPower_actionPerformed

`public void` **`botonPower_actionPerformed`**`(java.awt.event.ActionEvent e)`
Método para capturar el Action Event del botón Power del osciloscopio
Parameters: `e` - Action Event del botón Power

---

botonIntens_actionPerformed

`public void` **`botonIntens_actionPerformed`**`(java.awt.event.ActionEvent e)`
Método para capturar el Action Event del botón Intensidad del osciloscopio
Parameters: `e` - Action Event del botón Intensidad

---

botonFocus_actionPerformed

`public void` **`botonFocus_actionPerformed`**`(java.awt.event.ActionEvent e)`
Método para capturar el Action Event del botón Focus del osciloscopio
Parameters: `e` - Action Event del botón Focus

---

clearMatrizPuntos

`public void` **`clearMatrizPuntos`**`()`
Método para borrar la matriz de puntos de la pantalla del osciloscopio

Perifericos → Osciloscopio → OsciloscopioGUI

perifericos.osciloscopio

Class OsciloscopioGUI

public abstract class **OsciloscopioGUI**
extends JInternalFrame
Interfaz gráfico del osciloscopio
Title:**OSCILOSCOPIO GUI** Copyright (c) 2005

Detalles del Constructor

OsciloscopioGUI

`public` **`OsciloscopioGUI`**`()`

Constructora del interfaz gráfico de la clase Osciloscopio

Detalles de los métodos

jbInit

`private void` **`jbInit`**`()`

Método que inicia los componentes gráficos del interfaz

Throws: `Exception` –

Perifericos → Osciloscopio → OsciloscopioHilo

perifericos.osciloscopio

Class OsciloscopioHilo

public class **OsciloscopioHilo**
implements Runnable

Title:Simulador Motorola68K Copyright (c) 2005

Detalles del constructor

OsciloscopioHilo

`public` **`OsciloscopioHilo`**`(Osciloscopio o, boolean pv)`

Constructora de la clase hilo del osciloscopio

Parameters: `o` - osciloscopio
`pv` - primera vez

Detalles de los métodos

setPrimeraVez

`public void` **`setPrimeraVez`**`(boolean pv)`

Método que indica si ha sido la primera vez que se enciende el osciloscopio

Parameters: `pv` –

setSinSeñal

`public void` **`setSinSeñal`**`(boolean señal)`

Método para indicar si se envia señal al osciloscopio o no

Parameters: `señal` -

run

```
public void run()
```
Método run del hilo del osciloscopio

Perifericos → Osciloscopio → Pantalla

perifericos.osciloscopio

Clase Pantalla

public class **Pantalla**
extends javax.swing.JLabel
Clase para representar la pantalla del osciloscopio
Title:**PANTALLA** Copyright (c) 2005

Detalle del constructor

Pantalla

```
public Pantalla()
```
Constructora de la clase **Pantalla**

Detalles de los métodos

setOsciloscopio

```
public void setOsciloscopio(perifericos.osciloscopio.Osciloscopio os)
```
Método que actualiza el osciloscopio asociado a la pantalla
Parameters: `os` -

---

paint

```
public void paint(java.awt.Graphics imagen,boolean[][] matrizPuntos,
                  float intensidad)
```
Redefinimos el método paint de la pantalla
Parameters: `imagenPantalla` - imagen de la pantalla a dibujar
`matrizPuntos` - matriz de los puntos a dibujar 256x256
`intensidad` - intensidad de la imagen

---

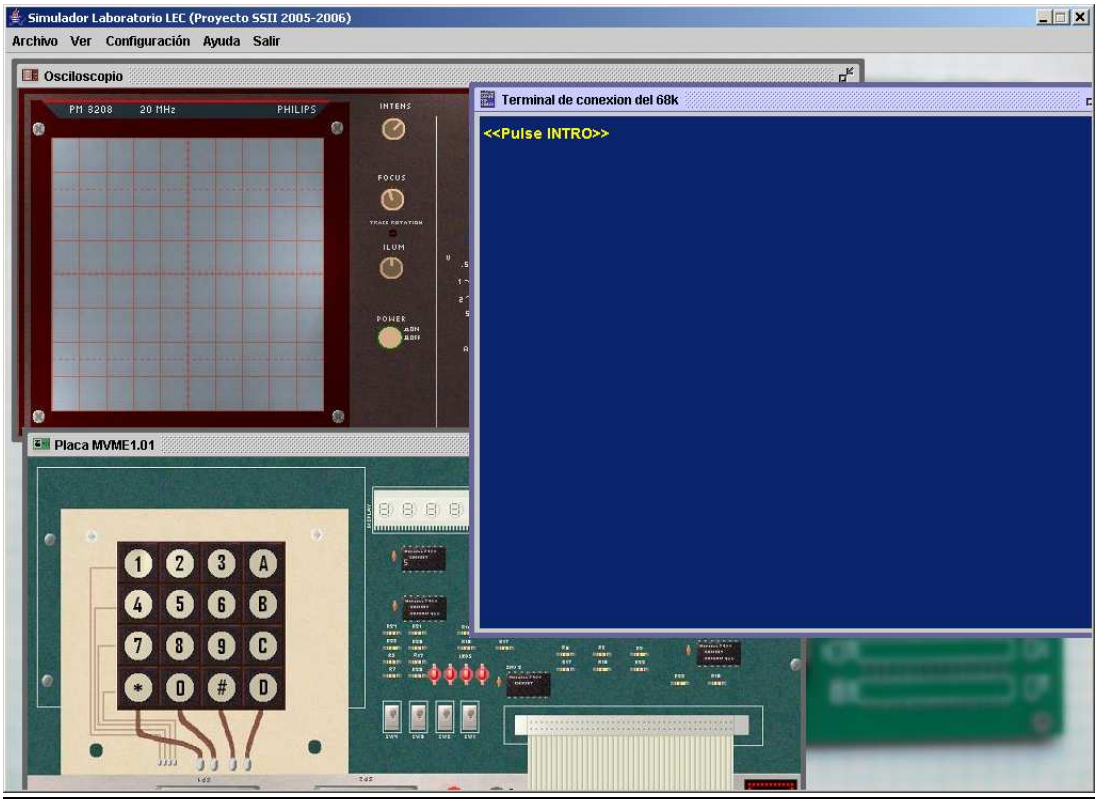
pantallaInicial

```
public void pantallaInicial(java.awt.Graphics imagen)
```
Método que inicializa la pantalla de arranque del osciloscopio
Parameters: `imagenPantalla` - imagen de la pantalla a mostrar

4. Utilización del modelo (Guía de uso)

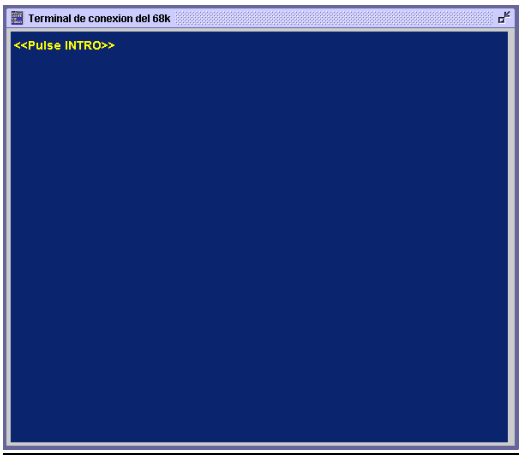
4.1 Introducción al entorno

Según empieza la ejecución del programa el usuario se encontrara la siguiente interfaz cuyo funcionamiento iremos analizando a continuación:



Consola

Este componente de la interfaz simula el comportamiento del terminal de conexión del 68000. Mediante ella el usuario podrá cargar sus prácticas, ejecutarlas y confirmar su correcto funcionamiento mediante trazas, o analizando el contenido de registros o posiciones de memoria:



En primer lugar, y antes de comenzar habrá que pulsar intro como en la consola real y a partir de aquí podremos aplicar las siguientes instrucciones (comentadas en mayor profundidad en el apartado de la consola):

- **.a** Muestra el contenido de todos los registros de direcciones. En caso de seleccionar uno de la forma **.ai** podremos verlo o en caso de añadir un valor hexadecimal detrás podremos modificarlo **.ai vv**
- **.d** Funciona de la misma manera que el apartado anterior, pero es referente a los registros de datos.
- **.pc** Muestra el estado del contador de programa o en caso de pongamos una dirección **.pc dddd** pondrá el pc en la posición de memoria que le asignemos.
- **.sr** Muestra el contenido del registro de condiciones.
- **br** Muestra los breakpoint existentes o anade uno si ponemos una direccion.
- **nobr** Quita todos los breakpoints o uno en concreto si damos una direccion.
- **lo1** Carga una practica previamente seleccionada en Archivo.
- **go** Ejecuta el programa desde la dirección actual del pc o en caso de que queramos que empiece la ejecución desde un punto en concreto añadiremos la dirección de comienzo de la forma go dddd.
- **md** Memory Display. Muestra el contenido de memoria de una posición solicitada. En caso de solicitarlo podemos determinar cuantas instrucciones queremos que muestre y si queremos o no que estas aparezcan desensambladas para que sean mas faciles detratar.
- **mm** Memori Modify. Permite modificar el contenido de posiciones de memoria.
- **tr** Ejecuta una instrucción en paso a paso ye entra en modo traza, con dar a enter ejecutara el siguiente paso automáticamente.

Osciloscopio

La parte que realmente nos interesa del osciloscopio es la pantalla en si, donde se mostrara la salida de algunas de las practicas de la asignatura destinadas a utilizar el conversor analógico-digital y los punteros de las coordenadas x e y del tubo de rayos catódicos del osciloscopio (DACA y DACB).

Los botones pulsables de esta interfaz se indican mediante la aparición de otro cursor (una manita) y son los que en el dibujo anterior aparecían rodeados con círculos verdes:

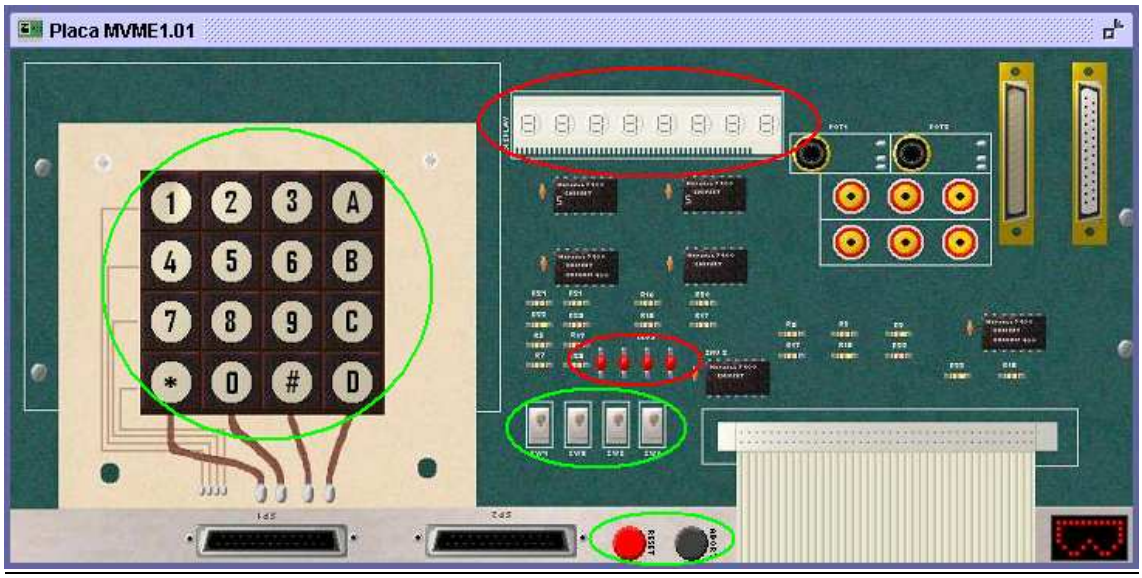
-Boton "Power":
Deberemos pulsar este botón si queremos utilizar y ver la salida de los programas mediante el osciloscopio, notaremos que el osciloscopio esta encendido ya que cambia la iluminación de la pantalla y aparece un punto moviéndose de lado a lado para indicar que esta esperando una señal de entrada. A partir de aquí podríamos ejecutar nuestro programa en ensamblador.

-Boton "focus":
Para hacer algo más real la interfaz podemos regular la intensidad del verde del puntero de rayos del osciloscopio, habiendo cuatro intensidades de iluminación para que el usuario elija con la que esté más cómodo. Al pulsar el botón, este irá rotando en proporción a la intensidad que representa.

-Boton "intens":
La explicación de la existencia de este botón es semejante a la del anterior y es meramente estética y para que el simulador sea todo lo real posible. En este caso podemos seleccionar entre cuatro iluminaciones distintas para poner de fondo a la pantalla.

Obs: Aunque los demás botones parezcan reales son únicamente parte de la interfaz y no hemos considerarlo implementar su funcionamiento ya que no aportaban nada a la asignatura.

Placa MVME 1.01



Marcamos en verde los botones pulsables y en rojo los componentes que nos van a mostrar la salida del sistema.

Entradas

1)Teclado hexadecimal:
Todos los botones del teclado son pulsables y modifican las líneas del teclado matricial pertinentes, con lo que el usuario tendrá que programar para capturar, tratar y definir lo que hace el pulsado de cada botón.

2)Switches
Los cuatro interruptores funcionan emitiendo señal de 1 cuando están abajo y 0 cuado están arriba.

3)Botones de reset y abort
Los botones de reset y abort están destinados a finalizar la ejecución de un programa en uso. Se diferencian en que el reset además limpia los registros del sistema. Estos botones no son programables y realizaran su función siempre independientemente de programa en ejecución.

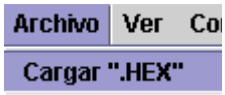
Salidas

1)Leds
Los cuatro leds tienen capacidad para encenderse o apagarse según programemos a 1 o a 0 la posición de memoria que les corresponde. Al comenzar la ejecución del simulador todos los leds aparecen encendidos y se apagarán con el cargado del primer programa.

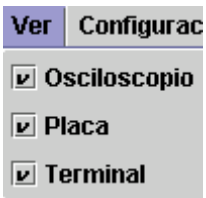
2)Displays
Los displays de 8 segmentos son totalmente programables tanto la selección de cuáles escriben como la selección de qué es lo que escriben.

Barra de herramientas (Opciones)

Archivo → Cargar hexadecimal: seleccionamos mediante el sistema de archivos la practica ya ensamblada que queremos probar.



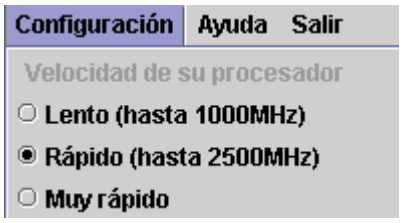
Ver: Nos permite seleccionar las ventanas del entorno que queremos tener activadas y visibles. En caso de minimizar aparecen forma de pestaña en la zona inferior izquierda.



Configuración → Muy rápido: Aumenta los retardos del simulador para conseguir una velocidad más parecida al aparato real funcionando sobre computadores "rápidos"que consideramos que son lo que operan a velocidades superiores.

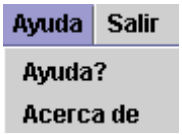
Configuración → Rápido: En este caso se usarán los retardos por defecto para ordenadores de gama media (esta es la configuración recomendada para usar en el laboratorio y es la que viene predefinida al ejecutar la aplicación)

Configuración → Lento: En caso de que no contemos con un ordenador muy potente nuestra aplicación intentará contrarrestarlo minimizando los retardos de temporización y los retardos entre pasos de ejecución, además de que cargara al computador con menos refrescos de la pantalla del osciloscopio. Con esto conseguiremos cargar con menos trabajo al sistema, para que incluso equipos antiguos puedan ejecutar el simulador sin problemas.



Ayuda → Para utilizar la ayuda se nos remite a la ayuda ofrecida por la consola, deberemos pulsar HE y luego buscar los temas que nos interesen.

Acerca de → Aparece la versión y los componentes del grupo que realizaron el simulador



Salir: Finaliza la ejecución del programa, al igual que el aspa, y solicita confirmación antes de salir.

4.2 Ejecución de una práctica

Los pasos a seguir para ejecutar una práctica son básicamente los mismos que seguimos cuando ejecutamos una práctica real en el laboratorio.

Observación: Sólo hay que decir que suponemos que el usuario ya tiene un programa en ensamblador compilado con el entorno del 68000 y que dicho código está en formato srecord (.HEX) sin errores. En caso de haber algún problema el simulador tiene capacidad para detectar errores en el checksun al igual que el sistema real, y notificarlo.

1) Selección. En primer lugar tendremos que seleccionar el hexadecimal de la práctica en archivo→ cargar

2**) Cargado**. Una vez hecho esto vamos a la consola y escribimos la instrucción de cargado **lo1**.Podremos utilizar la opción avanzada **lo1;x** para que al cargar veamos la transmisión del srecord que representa la información en hexadecimal referente al programa ensamblador y que va a ser volcada a la memoria simulada del sistema.

3) Ejecución. Podemos hacerlo de dos formas: fijando previamente el contador de programa con **.pc 2000** (la dirección de memoria 2000 es la referente al comienzo de programa en todas las practicas de la asignatura) y luego ejecutando un **go**, o directamente podríamos aplicar un **go 2000.**

4) Depuración. En caso de que el programa que hemos implementado no funcione como esperábamos tenemos a nuestra disposición todo el abanico de recursos que nos ofrecía el 68000 simulados. Podemos en cualquier momento realizar una traza con **tr** e ir viendo como se ejecuta paso a paso el programa. También podríamos consultar y modificar el estado de los registros con **.d** o **.a** o de posiciones de memoria mediante **mm** y **md**.

5. Conclusiones

Es por todos sabida la importancia que tiene en muchos sectores profesionales, la utilización de aplicaciones que permitan la simulación de cierto hardware. Por ejemplo, campos como la aeronáutica se han servido de herramientas para simuladores de vuelo, test de pruebas en automoción,... lo que les ha permitido un importante ahorro económico a parte de un incremento en la seguridad.

El uso de simuladores y emuladores actualmente es un campo muy importante de investigación en diversas empresas. Con nuestra aplicación hemos podido comprobar las ventajas que otorga disponer de una cómoda, portable y fácilmente ampliable herramienta software, librándonos de las incomodidades y elevado coste que supone el hardware.

Partiendo de nuestra principal motivación, nos sentimos bastante satisfechos al conseguir implementar un emulador del ***Motorola68k*** con los periféricos asociados a la MVME101 usados en los laboratorios, en un entorno multiplataforma Java. Sabemos que existen otros emuladores del ***Motorola68k***, aunque dificilmente con tal nivel de detalle y de periféricos disponibles como el realizado en esta aplicación.

En su momento, cuando los integrantes de este proyecto cursamos el laboratorio de LEC, vimos que al contrario que con otros laboratorios, no podiamos trabajar con nuestras prácticas en otro lugar que no fuera la facultad. Por tanto, el conseguir desarrollar una aplicación que permitiera salvar este obstáculo ha sido un importante logro, sobretodo en el plano pedagógico, como herramienta de trabajo para los demás alumnos de la facultad.

Además tambien cabe destacar que hemos conseguido familiarizarnos con la organización y funcionamiento de un, aunque ya algo anticuado, importante chip en la historia de la informática como es el ***Motorola68k***. El microprocesador ***Motorola68000*** fue lanzado al mercado en 1980 y fue el primero de una familia de microprocesadores, la *familia 68000* o *m68k* . Varias e importantes empresas dentro de la informática, han usado desde entonces los chips de la familia 68000, desde **Apple** en el ordenador personal **Lisa** de principios de los 80 y después en los primeros **Macintosh**, como **Texas Instruments** en sus calculadoras, **Sega** en su videoconsola ***Megadrive*** o **Commodore** en su gama ***Amiga***, hasta **Sun Microsystems** o **Silicon Grapchis** para sus estaciones de trabajo (antes de pasarse a los microprocesadores *SPARC* y *MIPS* respectivamente).

Estos nuevos conocimientos podrán ser importantes en nuestro futuro profesional, abriéndonos puertas como las de el desarrollo de emuladores de procesadores, al habernos iniciado ya en este terreno, conociendo algunos de los diversos problemas y situaciones presentes en la implementación y funcionamiento de un microprocesador.

6. Bibliografía

Para la realización de este proyecto utilizamos diversas fuentes de información y documentación, tanto por medio de manuales suministrados por el profesor como por medio de buscadores de Internet.

En lo referente al **Motorola68000**, para documentarnos acerca de los tipos, mnemotécnica, formatos, codificación binaria de las instrucciones, nos servimos principalmente del manual oficial de Motorola:

- **MOTOROLA M68000 FAMILY (Programmer's Referente Manual).**

También buscamos información en buscadores on-line (google) debido a diversos fallos encontrados en los formatos y codificación de algunas instrucciones. Por ejemplo encontramos información de utilidad en las siguientes páginas:

- http://www.geocities.com/SiliconValley/Way/7211/pag6.html
- http://www.fdi.ucm.es/profesor/horten

Sobre el sistema microcomputador y su Programa Monitor ***MVME101***, hicimos uso del manual de usuario:

- **MVME101bug Debug Package User's Manual. MICROSYSTEMS.**

Por último, los datos acerca del puesto de trabajo, información sobre la estructura física, organización y funcionamiento de la tarjeta de periféricos y de interfaces de E/S, los recopilamos de apuntes de la página de la Facultad de Informática sobre las asignaturas de arquitectura de computadores, principalmente LEC, de la cual utilizamos las prácticas de los integrantes del grupo o de otros alumnos que nos las cedieron para depurar y simular el comportamiento de la aplicación.